Abstract Acceleration in Linear Relation Analysis

Laure Gonnord, Peter Schrammel

▶ **To cite this version:**

Laure Gonnord, Peter Schrammel. Abstract Acceleration in Linear Relation Analysis. Science of Computer Programming, 2014, 93, part B (125 - 153), pp.125 - 153. 10.1016/j.scico.2013.09.016 . hal-00876627

HAL Id: hal-00876627
https://hal.science/hal-00876627

Submitted on 20 Oct 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Author Version of « Abstract Acceleration in Linear Relation Analysis » Science of Computer Programming, 2014.

Abstract Acceleration in Linear Relation Analysis[☆]

Laure Gonnord[a], Peter Schrammel[b]

[a]*University of Lille, LIFL, Cité scientifique - Bâtiment M3, 59655 Villeneuve d'Ascq Cedex, France,* laure.gonnord@lifl.fr

[b]*University of Oxford, Department of Computer Science, Wolfson Building, Parks Road, Oxford, OX1 3QD, UK,* peter.schrammel@cs.ox.ac.uk

Abstract

Linear relation analysis is a classical abstract interpretation based on an over-approximation of reachable numerical states of a program by convex polyhedra. Since it works with a lattice of infinite height, it makes use of a widening operator to enforce the convergence of fixed point computations. Abstract acceleration is a method that computes the precise abstract effect of loops wherever possible and uses widening in the general case. Thus, it improves both the precision and the efficiency of the analysis. This article gives a comprehensive tutorial on abstract acceleration: its origins in Presburger-based acceleration including new insights w.r.t. the linear accelerability of linear transformations, methods for simple and nested loops, recent extensions, tools and applications, and a detailed discussion of related methods and future perspectives.

Keywords: Program analysis, abstract interpretation, linear relation analysis, polyhedra, acceleration

1. Introduction

Linear relation analysis (LRA)[1, 2] is one of the very first applications of abstract interpretation [3]. It aims at computing an over-approximation of the reachable states of a numerical program as a convex polyhedron (or a set of such polyhedra). It was applied in various domains like program parallelization [4], automatic verification [5, 6], compile-time error detection [7], and invariant generation to aid automated program proofs [8, 9].

In comparison to interval or octagonal static analyses, polyhedral analyses are more expensive, but also more precise. However, precision is often compromised by the use of widenings that are required to guarantee convergence of the analysis. A lot of techniques have been proposed to improve widenings or to limit their effects, like delaying widening, widening with thresholds [10], landmarks [11] or guided static analysis [12], to mention just a few. Often these methods are based on heuristics and cure certain symptoms of the problem.

One may ask the question whether there are certain cases where widening can be performed precisely. This leads us to so-called *acceleration techniques*, proposed by several authors [13, 14,

[☆]This work has been partially supported by the APRON project of the "ACI Sécurité Informatique" of the French Ministry of Research, the INRIA large-scale initiative Synchronics, and the ARTEMIS VeTeSS project.

15, 16, 17] investigating the problem of finding subclasses of numerical programs for which the reachable set can be computed exactly. Roughly, these methods can handle loops with a restricted form of linear arithmetic in programs without nested loops such that the reachable states can be characterized in Presburger arithmetic. However, these computations have a very high complexity (doubly exponential) which limits the applicability of these methods in practice.

Inspired by these methods, Gonnord and Halbwachs proposed *abstract acceleration* [18] in LRA as a *complement to widening*. The rationale is to compute the precise *abstract* effect of a loop whenever possible, and otherwise to use widening. Hence, this method integrates seamlessly with classical linear relation analyses *without restricting the class of programs* that can be handled by the overall analysis, and without using expensive computations in Presburger arithmetic. Actually, the proposed abstract accelerations are very cheap and effectively speed up analysis because less iterations are required in comparison to a classical LRA.

Contributions and outline The goal of this article is to give a comprehensive tutorial on linear abstract acceleration. We cover the following aspects:

- Origins: We recall the principles of *linear relation analysis* and summarize the main results of *Presburger-based acceleration* (§2).
- Theoretical background: We develop the notion of *linear accelerability of linear transformations*, which, on the one hand, revisits results known from Presburger-based acceleration and, on the other hand, gives some new insights (§3).
- The method: We give an enhanced account of the *abstract acceleration* of simple loops (§4) including concise proofs and a new accelerable case. Then, we will show how these methods can be applied to multiple loops (§5).
- Extensions: We briefly summarize recent extensions of abstract acceleration to *reactive systems* and *backward analysis* (§6).
- Tools: We briefly describe the tools ASPIC and REAVER that implement these methods and give some experimental results (§7).
- Properties: We give a detailed *discussion* of its properties and related work (§8).
- Conclusion: Finally, we discuss some *open problems and future perspectives* (§9).

2. Preliminaries

As abstract acceleration combines linear relation analysis with acceleration, we will briefly recall the basic concepts of these two areas and introduce some notations used throughout the paper.

2.1. Linear Relation Analysis

The goal of LRA is to attach to each control point of a program a system of linear inequalities satisfied by the numerical variables whenever the control is at that point. This is done by propagating systems of linear inequalities along the control paths of the program.

2.1.1. Programs

We consider numerical programs represented by a *control flow graph* (sometimes called *interpreted automaton*):

Definition 1 (Control Flow Graph). *A control flow graph (CFG) $\langle \Sigma, L, \rightsquigarrow, X^0 \rangle$ is defined by*
- *the state space Σ with $\Sigma = \mathbb{R}^n$ (or $\mathbb{Z}^n$),*
- *a set of locations L,*
- *a set of transitions $\rightsquigarrow : L \times T \times L$, where transitions are labeled with a transition relations $\tau \in T = \wp(\Sigma^2)$, and*
- *$X^0 : L \to \wp(\Sigma)$ defines for each location the set of initial states.*

We will mostly consider transition functions $\tau : \boldsymbol{x} \mapsto \boldsymbol{x}'$ denoted as $G(\boldsymbol{x}) \to \boldsymbol{x}' = A(\boldsymbol{x})$, where the *guard* G is a conjunction of constraints over state variables $\boldsymbol{x}$ that must be satisfied in order to take the transition, and the *action* (or assignment) A that computes the next state $\boldsymbol{x}'$ as a function of the current state $\boldsymbol{x}$. The left-hand side of Fig. 1 gives an example of such a graph: locations are ℓ_1 and ℓ_2, the two arrows denote transitions, and the initial state for location ℓ_1 is $\{x_1 = 0, x_2 = 6\}$ and is empty for location ℓ_2.

Definition 2 (Operational Semantics). *An* execution *of a CFG is a sequence*

$$(\ell_0, \boldsymbol{x}_0) \to (\ell_1, \boldsymbol{x}_1) \to \dots (\ell_k, \boldsymbol{x}_k) \to \dots$$

such that $\boldsymbol{x}_0 \in X^0(\ell_0)$ and for any $k \geqslant 0 : \exists (\ell_k, \tau, \ell_{k+1}) \in \rightsquigarrow : \boldsymbol{x}_{k+1} = \tau(\boldsymbol{x}_k)$. where $\ell \in L$ and $\boldsymbol{x} \in \Sigma$.

Definition 3 (Collecting Semantics). *The collecting semantics defines the set of reachable states for all locations $X : L \to \Sigma$ as the least fixed point of the following equation:*

$$X = \lambda \ell'. X^0(\ell') \cup \bigcup_{(\ell, \tau, \ell') \in \rightsquigarrow} \tau(X(\ell))$$

where $\tau(X) = \{\boldsymbol{x}' \mid G(\boldsymbol{x}) \wedge \boldsymbol{x}' = A(\boldsymbol{x}) \wedge \boldsymbol{x} \in X\}$.
We will denote the right-hand side of the equation $F(X)$.

Mind that, by abuse of notation, τ may denote a transition function or its associated predicate transformer.

Since the collecting semantics is not computable in general, LRA based on abstract interpretation [3, 1] computes the corresponding abstract semantics over the abstract domain of convex polyhedra $Pol(\mathbb{R}^n)$. The abstraction is defined by the concretization (γ) and abstraction (α) functions that form the Galois connection $\wp(\mathbb{R}^n) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} Pol(\mathbb{R}^n)$ (see [3]).

2.1.2. Convex Polyhedra

A convex polyhedron can be either represented by
- the sets of *generators* (V, R), *i.e.*, the convex closure of vertices and rays

$$\gamma(V, R) = \{\boldsymbol{x} \mid \exists \boldsymbol{\lambda} \geqslant \boldsymbol{0}, \boldsymbol{\mu} \geqslant \boldsymbol{0} \ : \ \sum_i \lambda_i = 1 \wedge \boldsymbol{x} = \sum_i \boldsymbol{v}_i \lambda_i + \sum_j \boldsymbol{r}_j \mu_j\}$$

with the vertices $V = \{\boldsymbol{v}_1, \dots, \boldsymbol{v}_p\}$, $\boldsymbol{v}_i \in \mathbb{R}^n$ and the rays $R = \{\boldsymbol{r}_1, \dots, \boldsymbol{r}_q\}$, $\boldsymbol{r}_j \in \mathbb{R}^n$,

– or by a conjunction of linear *constraints* $\mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b}$, *i.e.*, an intersection of halfspaces

$$\gamma(\mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b}) = \{\boldsymbol{x} \mid \mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b}\}$$

with $\boldsymbol{x} \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$.

We can convert one representation into the other one using the double description method [19, 20], also known as *Chernikova's algorithm* [21, 22]. The set of convex polyhedra $Pol(\mathbb{R}^n)$ ordered by inclusion forms a lattice with $\top$ and $\bot$ denoting the polyhedra $\mathbb{R}^n$ and $\emptyset$ respectively.

Domain operations. Once a polyhedron is represented in both ways, some domain operations $(\sqcup, \sqcap, \ldots)$ can be performed more efficiently using the generator representation only, others based on the constraint representation, and some making use of both. A convex polyhedron X_1 is *included* in a convex polyhedron X_2 iff all vertices of X_1 belong to X_2, and all rays of X_1 are rays of X_2. A convex polyhedron is *empty* iff it has no vertex. The *intersection* (meet, $\sqcap$) of two convex polyhedra, computed by the conjunction of their constraint systems, is again a convex polyhedron: $X_1 \cap X_2 = X_1 \sqcap X_2$. In contrast, the *union* of two convex polyhedra is not convex in general. The domain operator is thus the convex hull (join, $\sqcup$) of the original polyhedra, computed by the union of their generators, which implies: $X_1 \cup X_2 \subseteq X_1 \sqcup X_2$. For the projection $\exists x_i : X$ *Fourier-Motzkin elimination* is used, which is an algorithm for eliminating variables from a system of linear inequalities, *i.e.*, the constraint representation (see, *e.g.*, [23] for details). The *Minkowski sum* of two polyhedra $X = X_1 + X_2$ is defined by $X = \{\boldsymbol{x}_1 + \boldsymbol{x}_2 \mid \boldsymbol{x}_1 \in X_1, \boldsymbol{x}_2 \in X_2\}$. The *time elapse* operation [5], defined as $X_1 \nearrow X_2 = \{\mathbf{x}_1 + t\mathbf{x}_2 \mid \mathbf{x}_1 \in X_1, \mathbf{x}_2 \in X_2, t \in \mathbb{R}^{\geqslant 0}\}$, can be implemented using the generator representations: $(V_1, R_1) \nearrow (V_2, R_2) = (V_1, R_1 \cup V_2 \cup R_2)$. The result of the *widening* operation $X_1 \nabla X_2$ consists, roughly speaking, of those constraints of X_1 that are satisfied by X_2 (see [2, 5] for a detailed presentation).

2.1.3. Analysis

The classical abstract interpretation-based reachability analysis [3, 24] employs the abstract domain operations above to over-approximate the fixed point of the abstract semantics equation $X = F(X)$ (cf. Def. 3). The analysis consists of three phases:

(1) an *ascending sequence* $(X_n)_{0\leqslant n\leqslant N}$ of applications of F:
$$X_0 = X^0 \qquad X_{n+1} = F(X_n) \quad \text{for } n < N$$
(2) a (delayed) *widening sequence* $(X'_n)_{0\leqslant n\leqslant N'}$ that is guaranteed to converge to a post-fixed point $X'_{N'}$ in a finite number of steps:
$$X'_0 = X_N \qquad X'_{n+1} = X'_n \nabla (F(X'_n)) \quad \text{until convergence } (X'_{N'} \sqsubseteq X'_{N'+1})$$
(3) a (truncated) *descending sequence* $(X''_n)_{0\leqslant n\leqslant N''}$ of applications of F for approaching the least fixed point:
$$X''_0 = X'_{N'} \qquad X''_{n+1} = F(X''_n) \quad \text{for } n < N''$$

Descending iterations generally do not converge: Cousot and Cousot [3] propose the use of a narrowing operator in order to force the convergence to a fixed point. In practice, the descending sequence is usually truncated, which is sound because the result of the widening sequence satisfies $F(X'_{N'}) \sqsubseteq X'_{N'}$, and hence, all elements of the descending sequence are upper bounds of the least fixed point.

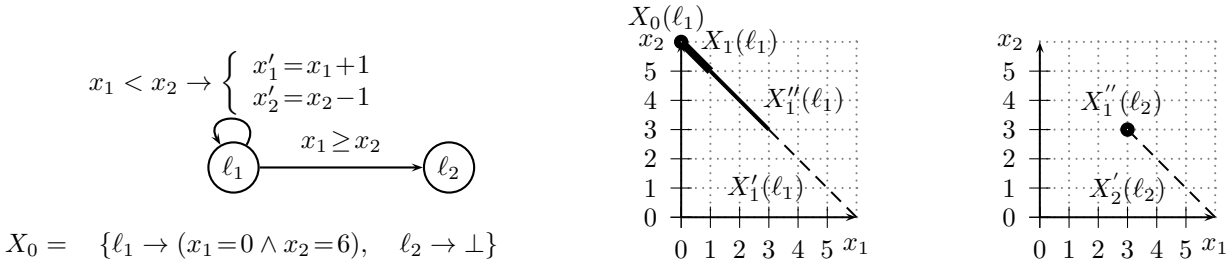


Figure 1: LRA Example 1: CFG with initial state X_0 (left), polyhedra during the analysis in ℓ_1 (center) and ℓ_2 (right). The dashed lines indicate the abstract value after widening.

Example 1 (Kleene Iteration with widening). *We analyze the CFG of the program depicted in Fig. 1 with $N = 1$ and $N'' = 1$:*

$$\begin{array}{rcl} X_0 & = & \{\ell_1 \to (x_1 = 0 \wedge x_2 = 6), \ell_2 \to \bot\} \\ X_1 & = & \{\ell_1 \to (0 \leqslant x_1 \leqslant 1 \wedge x_1 + x_2 = 6), \ell_2 \to \bot\} \end{array}$$

$$\begin{array}{rcl} X_1' & = & \{\ell_1 \to (0 \leqslant x_1 \leqslant 1 \wedge x_1 + x_2 = 6) \nabla (0 \leqslant x_1 \leqslant 2 \wedge x_1 + x_2 = 6), \ell_2 \to \bot \nabla \bot\} = \\ & & \{\ell_1 \to (0 \leqslant x_1 \wedge x_1 + x_2 = 6), \ell_2 \to \bot\} \\ X_2' & = & \{\ell_1 \to (0 \leqslant x_1 \wedge x_1 + x_2 = 6) \nabla (0 \leqslant x_1 \wedge x_1 + x_2 = 6), \ell_2 \to \bot \nabla (3 \leqslant x_1 \wedge x_1 + x_2 = 6)\} = \\ & & \{\ell_1 \to (0 \leqslant x_1 \wedge x_1 + x_2 = 6), \ell_2 \to (3 \leqslant x_1 \wedge x_1 + x_2 = 6)\} = X_3' = X_0'' \\ X_1'' & = & \{\ell_1 \to (0 \leqslant x_1 \leqslant 3 \wedge x_1 + x_2 = 6), \ell_2 \to x_1 = x_2 = 3\} \end{array}$$

In practice, abstract interpreters compute the fixed point following an iteration strategy that take into account the transitions in a specific order (cf. [25]), and widening is only applied at the loop heads.

Improvements of the Widening Operator. Although Cousot and Cousot [24] show that the approach using Kleene iteration with widening and infinite height lattices can discover invariants that finite height lattices cannot discover, the dynamic approximations induced by widening lead quite often to an important loss of precision. There are several reasons for these problems:

– The standard widening operators are *not monotonic*, *e.g.*, $[0,2] \nabla [0,4] = [0,\infty]$, but $[1,2] \nabla [0,3] = \top$ (although $[1,2] \sqsubseteq [0,2]$ and $[0,3] \sqsubseteq [0,4]$).
– Descending iterations fail to recover information if the result of the widening sequence is already a fixed point, *i.e.*, $F(X'_{N'}) = X'_{N'}$.

Numerous improvements of the widening operators and modified iteration strategies have been proposed. We refer to, *e.g.*, [26, 27, 28] for surveys of such improvements, some of them will be discussed in the related work (§8). In this line of research, abstract acceleration can be viewed as a precise, monotonic widening operator applicable to a specific class of loops.

2.2. Presburger-Based Acceleration

Acceleration methods aim at computing the *exact* set of reachable states in numerical transition systems. They are motivated by the analysis of communication protocols often modeled using

counter machines or Petri nets. First achievements date back to the 1990s [13, 29, 30, 15, 31]. Unlike abstract interpretation, which overcomes the undecidability issue by computing a conservative approximation, acceleration identifies classes of systems (of a certain structure and with certain transition relations) for which the reachability problem is decidable. Acceleration methods have been shown to be applicable to a variety of systems including pushdown systems and systems of FIFO channels for instance (cf. [32]). In this section, we summarize the main results w.r.t. the acceleration of counter systems, which is based on Presburger arithmetic.

Presburger arithmetic. Presburger arithmetic [33] is the first-order additive theory over integers $\langle \mathbb{Z}, \leqslant, + \rangle$. Satisfiability and validity are decidable in this theory. A set is *Presburger-definable* if it can be described by a Presburger formula. For example, the set of odd natural numbers x can be defined by the Presburger formula $\exists k \geqslant 0 : x = 1 + 2k$, whereas for example the formula $\exists k : y = k \cdot k$ characterizing the quadratic numbers y is not Presburger because of the multiplication of variables.

Counter systems. Counter systems are a subclass of the program model defined in Def. 1 with $\Sigma = \mathbb{Z}^n$, and initial states X^0 and transition relations $R(\boldsymbol{x}, \boldsymbol{x}') \in \wp(\Sigma^2)$ defined by Presburger formulas. Counter systems generalize Minsky machines [34], thus the reachability problem is undecidable. In general, the reachable set of a counter system is not Presburger-definable [35] because of the following two reasons:

(1) The reflexive and transitive closure R^* is not always Presburger-definable.
(2) In the case of a system with nested loops where the reflexive and transitive closures R^* of all circuits in the system are Presburger-definable, the reachable set of the whole system is not Presburger-definable in general, because there are infinitely many possible sequences of these circuits.

Issue (1) is addressed by identifying a class of *accelerable* relations R, *i.e.*, for which the transitive closure R^* is Presburger-definable:

Definition 4 (Presburger-linear relations with finite monoid). *The transition relation* $R(\boldsymbol{x}, \boldsymbol{x}') = (\varphi(\boldsymbol{x}) \wedge \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d})$ *is Presburger-linear with finite monoid iff* φ *is a Presburger formula and* $\langle C^*, \cdot \rangle$ *is a finite, multiplicative monoid, i.e., the set* $C^* = \{\mathbf{C}^k \mid k \geqslant 0\}$ *is finite.*

Theorem 1 (Presburger-definable transitive closure [31, 36]). *If R is a Presburger-linear relation with finite monoid, then R^* is Presburger-definable.*

The finiteness of the monoid is polynomially decidable [31]. The tool LASH [31, 14] implements these results.

Example 2 (Translation). *An example of transition relations of which the transitive closure is Presburger-definable are* translations*: $R(\boldsymbol{x}, \boldsymbol{x}') = (\varphi(\boldsymbol{x}) \wedge \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d})$. The variables are translated in each iteration by a constant vector $\boldsymbol{d}$. A translation is trivially finite monoid because $\mathbf{C} = \mathbf{I}$. The transitive closure is given by the Presburger formula:*

$$R^*(\boldsymbol{x}, \boldsymbol{x}') = \exists k \geqslant 0 : \boldsymbol{x}' = \boldsymbol{x} + k\boldsymbol{d} \wedge \forall k' \in [0, k-1] : \varphi(\boldsymbol{x} + k'\boldsymbol{d})$$

Issue (2) is adressed by the concept of flat acceleration:

Flat systems. A system is called *flat* if it has no nested loops, or more precisely if any location of the system is contained in at most one elementary cycle of the system [17]. This notion allows us to identify a class of systems for which the set of reachable states can be computed exactly:

Theorem 2 (Presburger-definable flat systems [36]). *The reachability set of a counter system is Presburger-definable if the system is flat and all its transitions are Presburger-linear relations with finite monoid.*

Although there are many practical examples of flat systems [37], most systems are non-flat.

Application to non-flat systems. The idea of Finkel et al [36, 32] is to partially unfold the outer loops (circuits) of nested loops in order to obtain a flat system that is simulated by the original system. Such a system is called a *flattening*. The algorithm is based on heuristically selecting circuits of increasing length and enumerating flattenings of these circuits. Hence, the algorithm terminates in case the system is *flattable*, *i.e.*, at least one of its (finite) flattenings has the same reachability set as the original system. However, flattability is undecidable [32]. All these techniques are implemented in the tool FAST [17, 38, 39, 32, 40].

3. Linear Accelerability of Linear Transformations

Similarly to Presburger-based acceleration, the abstract acceleration methods that we are going to present in §§4–6 target loops with linear transformations. To this purpose, we revisit the notion of linearly accelerable linear transformations, and we show that this class is actually larger than the "Presburger-based linear relations with finite monoid" originally considered by Presburger-based acceleration.

3.1. Characterizing Linearly Accelerable Linear Transformations

The basic idea of Presburger-based acceleration is to identify and characterize the class of linearly (*i.e.*, using Presburger arithmetic) accelerable transitions. Presburger arithmetic is able to define sets that are conjunctions of linear inequalities (*i.e.*, polyhedra, $\mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b}$), congruences ($\exists k : x' = x + kd$), and finite unions ($\bigcup$) thereof. This gives us the following definition of linear accelerability:

Definition 5 (Linear accelerability). *A transition $\tau : \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$ is linearly accelerable iff its reflexive and transitive closure τ^* can be written as a finite union of sets*

$$\tau^* = \lambda X . \bigcup_l \{\widehat{\mathbf{C}}_l \boldsymbol{x} + k\widehat{\boldsymbol{d}}_l \mid k \geqslant 0, \boldsymbol{x} \in X\}$$

The classical accelerability ("finite monoid") criterion is merely based on the matrix $\mathbf{C}$. To elucidate the role of correlations between the coefficients of $\mathbf{C}$, $\boldsymbol{d}$ and the initial set X, we will give an alternative characterization based on the *homogeneous* form of affine transformations: any affine transformation of dimension n can be written as a linear transformation $\begin{pmatrix} \boldsymbol{x}' \\ x'_{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{C} & \boldsymbol{d} \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} \boldsymbol{x} \\ x_{n+1} \end{pmatrix}$ of dimension $n+1$ and with $x_{n+1} = 1$ in the initial set X.

We consider the *Jordan normal form*[1] $\overline{\mathbf{J}} \in \mathbb{C}^n$ of $\mathbf{C}' = \begin{pmatrix} \mathbf{C} & \boldsymbol{d} \\ \mathbf{0} & 1 \end{pmatrix} \in \mathbb{R}^n$ which can be obtained by a similarity transformation $\overline{\mathbf{J}} = \mathbf{Q}^{-1}\mathbf{C}'\mathbf{Q}$ with a nonsingular matrix $\mathbf{Q} \in \mathbb{C}^n$. $\overline{\mathbf{J}}$ is a block diagonal matrix consisting of Jordan blocks $\mathbf{J}_i$ associated with the eigenvalues $\lambda_i \in \mathbb{C}$ of $\mathbf{C}'$. Furthermore, we have $\overline{\mathbf{J}}^k = \mathbf{Q}^{-1}\mathbf{C}'^k\mathbf{Q}$ and thus: $\overline{\mathbf{J}}^k = \begin{pmatrix} \mathbf{J}_1^k & \cdots & \mathbf{0} \\ \cdots & \ddots & \cdots \\ \mathbf{0} & \cdots & \mathbf{J}_j^k \end{pmatrix}$.

We will now examine the linear accelerability of a Jordan block w.r.t. its size and associated eigenvalue[2]. We give the acceleration criteria and illustrate them with examples. For the detailed proofs, we refer to the Appendix A.1.

Lemma 1 (Jordan block of size 1). *A transition $\tau(X) : \boldsymbol{x}' = \mathbf{J}\boldsymbol{x}$ where $\mathbf{J}$ is a Jordan block of size 1 is linearly accelerable iff its associated eigenvalue is either zero or a complex root of unity, i.e., $\lambda \in \{0\} \cup \{e^{i2\pi\frac{q}{p}} \mid p,q \in \mathbb{N}\}$.*

We give some examples for such loops and their accelerations:

(E1) $\lambda = 0$: $X = \{1\}$, $\tau : x' = 0$: $\tau^*(X) = X \cup \{0\} = \{0,1\}$

(E2) $\lambda = e^{i2\pi\frac{1}{2}} = -1$: $X = \{1\}$, $\tau : x' = -x$: $\tau^*(X) = \bigcup_{0 \leqslant l \leqslant p-1} \{(-1)^l x \mid x \in X\} = \{-1,1\}$

(E3) $\lambda_{1,2} = e^{i2\pi\frac{1}{4}} = \pm i$: $X = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\}$, $\tau : \begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$:

$$\tau^*(X) = \bigcup_{0 \leqslant l \leqslant p-1} \left\{ \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}^l \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \;\middle|\; \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in X \right\} = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right\}$$

(In real-valued transformation matrices, complex eigenvalues are always conjugate.)

Lemma 2 (Jordan block of size 2). *A transition $\tau(X) : \boldsymbol{x}' = \mathbf{J}\boldsymbol{x}$ where $\mathbf{J}$ is a Jordan block of size 2 is linearly accelerable iff its associated eigenvalue is*

- *either zero ($\lambda = 0$) or*
- *a complex root of unity ($\lambda \in \{e^{i2\pi\frac{q}{p}} \mid p,q \in \mathbb{N}\}$) and if in this case the variable associated to the second dimension of the block has only a finite number of values in X.*

We give some examples:

(E4) $\lambda = 1$: $X = \left\{ \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$, $\tau : \begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$:

$$\tau^*(X) = \left\{ \begin{pmatrix} 1 & 3 \\ 0 & 1 \end{pmatrix}^k \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \;\middle|\; k \geqslant 0, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in X \right\} = \left\{ \begin{pmatrix} 3k \\ 1 \end{pmatrix} \mid k \geqslant 0 \right\}$$

(E5) $\lambda_{1,2} = e^{i2\pi\frac{1}{4}} = \pm i$: $\tau : \boldsymbol{x}' = \begin{pmatrix} 0 & -1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \boldsymbol{x}$ (see Appendix A.1 regarding its acceleration)

[1] We refer to textbooks in linear algebra and matrix theory, *e.g.*, [41].

[2] Boigelot's proof [31] for the finite monid criterion proceeds similarly, but he considers $\mathbf{C}$ instead of $\mathbf{C}'$.

Lemma 3 (Jordan block of size >2). *A transition $\tau(X) : \boldsymbol{x}' = \mathbf{J}\boldsymbol{x}$ where $\mathbf{J}$ is a Jordan block of size $m>2$ is linearly accelerable iff its associated eigenvalue is zero.*

We give an example for $m=3$:

$$\text{(E6)}\quad \lambda=0\text{: } X=\left\{\begin{pmatrix}1\\2\\3\end{pmatrix}\right\},\ \tau:\boldsymbol{x}'=\begin{pmatrix}2&2&-2\\5&1&-3\\1&5&-3\end{pmatrix}\boldsymbol{x}$$

$$\tau^*(X)=\bigcup_{0\leqslant l\leqslant m}\left\{\begin{pmatrix}2&2&-2\\5&1&-3\\1&5&-3\end{pmatrix}^l\boldsymbol{x}\;\middle|\;\boldsymbol{x}\in X\right\}=\left\{\begin{pmatrix}1\\2\\3\end{pmatrix},\begin{pmatrix}0\\2\\-2\end{pmatrix},\begin{pmatrix}-8\\-8\\16\end{pmatrix},\begin{pmatrix}0\\0\\0\end{pmatrix}\right\}$$

We summarize these results in the following theorem:

Theorem 3 (Accelerable linear transformations). *A linear transformation $\tau(X) : \boldsymbol{x}' = \mathbf{C}'\boldsymbol{x}$ is linearly accelerable iff its Jordan form consists of Jordan blocks $\mathbf{J}$ satisfying the following criteria:*

- *$\mathbf{J}$ is of size 1 and its associated eigenvalue $\lambda \in \{0\} \cup \{e^{i2\pi\frac{q}{p}} \mid p,q \in \mathbb{N}\}$.*
- *$\mathbf{J}$ is of size 2 and its associated eigenvalue $\lambda = 0$ or $\lambda \in \{e^{i2\pi\frac{q}{p}} \mid p,q \in \mathbb{N}\}$, and in the latter case the variable associated with the second dimension of the block has only a finite number of values in X in the Jordan basis.*
- *$\mathbf{J}$ is of size greater than 2 and its associated eigenvalue $\lambda = 0$.*

3.2. Comparison with Finite Monoid Acceleration

The background of the "finite monoid" criterion of Def. 4 is the following characterization of Boigelot (Theorem 8.53, [31]): $\boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$ is accelerable if $\exists q > 0$ such that $\mathbf{C}^q$ is diagonizable and all its eigenvalues are in $\{0,1\}$. In other words: the eigenvalues are either zero or roots of unity and all Jordan blocks of non-zero eigenvalues have size 1.

Theorem 4 (Jordan normal form of finite monoid affine transformations). *The Jordan form of the homogeneous transformation matrix $\begin{pmatrix}\mathbf{C}&\boldsymbol{d}\\\mathbf{0}&1\end{pmatrix}$, where $\{\mathbf{C}^k \mid k\geqslant 0\}$ is finite, consists of*

- *Jordan blocks of size 1 with eigenvalues which are complex roots of unity,*
- *at most one block of size 2 with eigenvalue 1 where the variable associated with the second dimension is a constant equal to 1, and*
- *blocks with eigenvalue 0 of any size.*

PROOF: See Appendix A.2. □

Hence, finite monoid transformations are strictly included in the characterization of accelerable linear transformations of Thm. 3.

The Jordan normal form gives an intuitive geometric understanding of the linear transformations. For finite monoid transformations, the Jordan form of the homogeneous transformation matrix $\overline{\mathbf{J}}$ is the direct product of

- Identity blocks: $\lambda=1$, size 1, *i.e.*, $\tau : x'=x$;
- Translation blocks (Ex. E4 with $x_2=1$): $\lambda=1$, size 2, second dimension equals 1 in X;
- Nilpotent ("projection") blocks (E1, E6): $\lambda=0$, size m ($\mathbf{J}^m=\mathbf{0}$);

- Rotation blocks (E2, E3): $\lambda = e^{i2\pi\frac{q}{p}}$, size 1: this corresponds to a rotation by $\theta = 2\pi\frac{q}{p}$.

We now relate the ($(n+1)$-dimensional) $\overline{\mathbf{J}}$ to the characterization of typical finite monoid transitions by the (ultimate) periodicity property of the original matrix $\mathbf{C}$, *i.e.*, $\mathbf{C}^{p+l} = \mathbf{C}^p$, $p \geqslant 0, l > 0$:

- Translations ($p = 0, l = 1, \boldsymbol{d} \neq \mathbf{0}$): $\overline{\mathbf{J}}$ consists of one translation block and $n-1$ identity blocks.
- Translations with resets ($p = 1, l = 1$, m reset variables, $n-m$ translations): $\overline{\mathbf{J}}$ consists of m nilpotent blocks of size 1, one translation block, and $n-m-1$ identity blocks.
- Purely periodic $\mathbf{C}$ ($p = 0$): $\overline{\mathbf{J}}$ consists of rotation blocks and zero or one translation blocks (depending on $\boldsymbol{d}$, cf. Appendix A.2).
- Ultimately periodic $\mathbf{C}$: $\overline{\mathbf{J}}$ consists of rotation blocks, nilpotent blocks with maximum size p, and zero or one translation blocks (depending on $\boldsymbol{d}$, cf. Appendix A.2).

The criterion of Thm. 3 identifies furthermore those transformations as accelerable where the eigenvalue is a root of unity and the second dimension of the block has a finite set of values in the initial set X. In addition to the above types of blocks, we have:

- Finite-cardinality translation blocks (E4 with a different X: see Ex. 3 below): $\lambda = 1$, size 2, second dimension has a finite number of values in X;
- Finite-cardinality translation-rotation blocks (E5: see [42] for a detailed example): $\lambda = e^{i2\pi\frac{q}{p}}$, size 2, second dimension has a finite number of values in X.

Note that, in the latter two cases, the application of τ^* to a Presburger formula describing the set X requires a check whether X is finite. Hence, τ^* itself cannot be written as a Presburger formula in general, but only $\tau^*(X)$.

Example 3 (Finite-cardinality translation). *The following loop translates x_1 by x_2 while x_2 remains unmodified: $\tau : x_1 + 2x_2 \leqslant 6 \rightarrow \boldsymbol{x}' = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \boldsymbol{x}$ with $X_0 = (x_1 = 0 \wedge x_2 \in \{2,3,4\})$. We can exactly accelerate the loop τ by enumerating the values of x_2 in X_0 and translating x_1 by each of these values:*

$$\begin{aligned} \tau^*(X_0) = \; & \{(x_1', 2) \mid \exists k \geqslant 0 : x_1' = x_1 + 2k \wedge X_0(\boldsymbol{x}) \wedge \forall 0 \leqslant k' < k : x_1' \leqslant 2\} \cup \\ & \{(x_1', 3) \mid \exists k \geqslant 0 : x_1' = x_1 + 3k \wedge X_0(\boldsymbol{x}) \wedge \forall 0 \leqslant k' < k : x_1' \leqslant 0\} \cup \\ & \{(x_1', 4) \mid \exists k \geqslant 0 : x_1' = x_1 + 4k \wedge X_0(\boldsymbol{x}) \wedge \forall 0 \leqslant k' < k : x_1' \leqslant -2\} \\ = \; & \{(0,2), (2,2), (4,2), (0,3), (3,3), (6,3), (0,4)\} \end{aligned}$$

The resulting points are depicted in Fig. 2.

4. Abstract Acceleration of Simple Loops

Abstract acceleration introduced by Gonnord and Halbwachs [18] reformulates acceleration concepts within an abstract interpretation approach: it aims at computing the best correct approximation of the effect of loops in the abstract domain of convex polyhedra.

The objective of abstract acceleration is to over-approximate the set $\tau^*(X), X \subseteq \mathbb{R}^n$ by a (single) convex polyhedron $\gamma(\tau^{\otimes}(\alpha(X))) \supseteq \tau^*(X)$ that is "close" to the convex hull of the exact set. Ab-

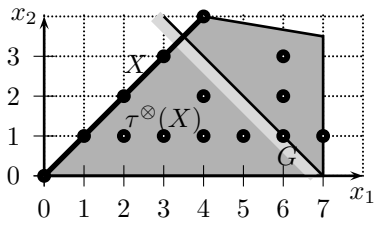


Figure 2: Finite-cardinality translation: $\tau : x_1+x_2 \leqslant 7 \;\rightarrow\; x'_1 = x_1+x_2 \;\wedge\; x'_2 = x_2$ with $X_0 = (0 \leqslant x_1 = x_2 \leqslant 4)$: reachable points and polyhedron computed by abstract acceleration (see §4.4).

stract acceleration targets affine transition relations $\tau : G \rightarrow A$ (actually functions) with polyhedral guards[3] $G = (\mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b})$ and affine actions $A = (\boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d})$.

Forgetting about the guard G for a moment, we can intuitively derive a notion of "abstract linear accelerability" by replacing the operators by their abstract counterparts in Def. 5: $\tau^{\otimes} = \lambda X. \bigsqcup_l \left(\widehat{\mathbf{C}}_l X \nearrow \{\widehat{\boldsymbol{d}}_l\}\right)$. This view elucidates the basic approximations made by abstract accelerations:

- Unions $\bigcup$ are approximated by the convex hull $\bigsqcup$.
- Congruences ($\exists k \geqslant 0 : \boldsymbol{x}' = \boldsymbol{x} + k\boldsymbol{b}$) are lost by the dense approximation using the $\nearrow$ operator.

We will first give abstract accelerations for the case of translations ($\mathbf{C} = \mathbf{I}$) and translations/resets ($\mathbf{C} = diag(c_1 \ldots c_n), c_i \in \{0,1\}$), and in a second step we will reduce the case of (general) finite monoid transitions to these two.

4.1. Translations

Theorem 5 (Translations). *Let τ be a translation $G \rightarrow \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d}$, then for every convex polyhedron X, the convex polyhedron*

$$\tau^{\otimes}(X) = X \sqcup \tau\big((X \sqcap G) \nearrow \{\boldsymbol{d}\}\big)$$

is a convex over-approximation of $\tau^*(X)$.

PROOF: $\boldsymbol{x}' \in \bigcup_{k \geqslant 1} \tau^k(X) \iff \boldsymbol{x}' \in \tau(\bigcup_{k \geqslant 0} \tau^k(X))$
$\iff \exists k \geqslant 0, \exists \boldsymbol{x}_0 \in X, \exists \boldsymbol{x}_k : \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \wedge \boldsymbol{x}_k = \boldsymbol{x}_0 + k\boldsymbol{d} \wedge G(\boldsymbol{x}_0) \;\wedge\; \forall k' \in [1,k] \;:\; G(\boldsymbol{x}_0 + k'\boldsymbol{d})$
$\iff \exists k \geqslant 0, \exists \boldsymbol{x}_0 \in X, \exists \boldsymbol{x}_k : \; \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \;\wedge\; \boldsymbol{x}_k = \boldsymbol{x}_0 + k\boldsymbol{d} \;\wedge\; G(\boldsymbol{x}_0) \;\wedge\; G(\boldsymbol{x}_k)$
(because G is convex)
$\Longrightarrow \exists \alpha \geqslant 0, \exists \boldsymbol{x}_0 \in X, \exists \boldsymbol{x}_k : \; \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \;\wedge\; \boldsymbol{x}_k = \boldsymbol{x}_0 + \alpha\boldsymbol{d} \;\wedge\; G(\boldsymbol{x}_0)$
(dense approximation; $G(\boldsymbol{x}_k)$ implied by $\boldsymbol{x}' \in \tau(\boldsymbol{x}_k)$)
$\iff \exists \boldsymbol{x}_0 \in X \sqcap G, \exists \boldsymbol{x}_k \;:\; \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \;\wedge\; \boldsymbol{x}_k \in (\{\boldsymbol{x}_0\} \nearrow \{\boldsymbol{d}\})$
$\iff \boldsymbol{x}' \in \tau((X \sqcap G) \nearrow \{\boldsymbol{d}\})$ □

Ideally, $\tau^{\otimes}(X)$ as defined in Thm. 5 should be the best over-approximation of $\tau^*(X)$ by a convex polyhedron. This is not the case as shown by the following example in one dimension. Let

[3] We will use the same notation for polyhedra X interexchangeably for both the predicate $X(\boldsymbol{x}) = (\mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b})$ and the set $X = \{\boldsymbol{x} \mid \mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b}\}$.

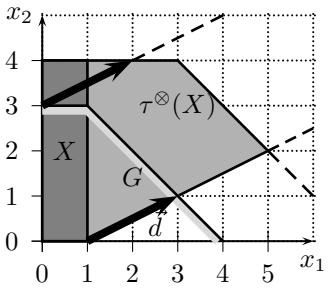


Figure 3: Abstract acceleration of a *translation* (Ex. 4) by vector $\boldsymbol{d}$ starting from X (dark gray) resulting in $\tau^{\otimes}(X)$ (whole shaded area).

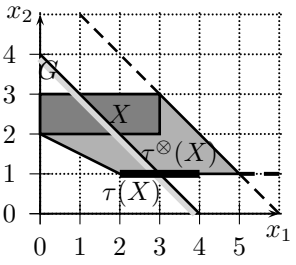


Figure 4: Abstract acceleration of *translations/resets* (Ex. 5) starting from X (dark gray): $\tau(X)$ (bold line) and result $\tau^{\otimes}(X)$ (whole shaded area).

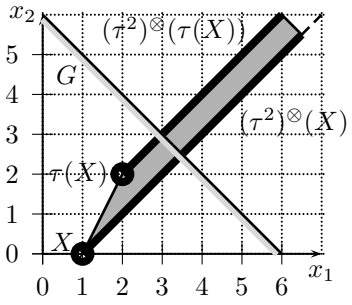


Figure 5: Abstract acceleration of a *finite monoid* (Ex. 6) starting from $X = \{(1,0)\}$, $\tau(X) = \{(2,2)\}$, $(\tau^2)^{\otimes}(X)$ and $(\tau^2)^{\otimes}(\tau(X))$ (bold lines), and the result $\tau^{\otimes}(X)$ (whole shaded area).

$X = [1,1]$ and $\tau : x_1 \leqslant 4 \to x_1' = x_1 + 2$. $\tau^{\otimes}(X) = [1,6]$, whereas the best convex over-approximation of $\tau^*(X) = \{1,3,5\}$ is the interval $[1,5]$. This is because the operations involved in the definition of $\tau^{\otimes}(X)$ manipulate dense sets and do not take into account arithmetic congruences. Considering the proof, this approximation takes place in the line ($\Rightarrow$) where the integer coefficient $k \geqslant 0$ is replaced by a real coefficient $\alpha \geqslant 0$.

Example 4 (Translation). *(see Fig. 3)* $\tau : \underbrace{x_1 + x_2 \leqslant 4 \ \wedge \ x_2 \leqslant 3}_{G} \to \begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \underbrace{\begin{pmatrix} 2 \\ 1 \end{pmatrix}}_{\boldsymbol{d}}$

Starting from $X = (0 \leqslant x_1 \leqslant 1 \ \wedge \ 0 \leqslant x_2 \leqslant 4)$ *we compute* $\tau^{\otimes}(X)$*:*

$$\begin{array}{rcl} X \sqcap G & = & (0 \leqslant x_1 \leqslant 1 \ \wedge \ 0 \leqslant x_2 \leqslant 3) \\ (X \sqcap G) \nearrow \{\boldsymbol{d}\} & = & (x_1 \geqslant 0 \ \wedge \ x_2 \geqslant 0 \ \wedge \ x_1 - 2x_2 \geqslant -6 \ \wedge \ -x_1 + 2x_2 \geqslant -1) \\ \tau((X \sqcap G) \nearrow \{\boldsymbol{d}\}) & = & \left\{ \begin{array}{l} x_1 \geqslant 0 \ \wedge \ 0 \leqslant x_2 \leqslant 4 \ \wedge \ x_1 - 2x_2 \geqslant -6 \ \wedge \\ -x_1 + 2x_2 \geqslant -1 \ \wedge \ x_1 + x_2 \leqslant 7 \end{array} \right. \\ \tau^{\otimes}(X) & = & (x_1 \geqslant 0 \ \wedge \ 0 \leqslant x_2 \leqslant 4 \ \wedge \ -x_1 + 2x_2 \geqslant -1 \ \wedge \ x_1 + x_2 \leqslant 7) \end{array}$$

4.2. Translations/Resets

Theorem 6 (Translations/resets). *Let τ be a translation with resets $G \to \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$, then for every convex polyhedron X, the convex polyhedron*

$$\tau^{\otimes}(X) = X \sqcup \tau(X) \sqcup \tau\big((\tau(X) \sqcap G) \nearrow \{\mathbf{C}\boldsymbol{d}\}\big)$$

is a convex over-approximation of $\tau^*(X)$.

Intuitively, Thm. 6 exploits the property that a translation with resets to *constants* (C is idempotent) iterated N times is equivalent to the same translation with resets, followed by a pure translation iterated $N-1$ times. Hence the structure of the obtained formula.

PROOF: The formula is trivially correct for 0 or 1 iterations of the self-loop τ. Using $C^k = C$, a recurrence immediately gives $\tau^k(\mathbf{C}\boldsymbol{x}_0 + \boldsymbol{d}) = (C\boldsymbol{x}_0 + \boldsymbol{d}) + kC\boldsymbol{d}$. It remains to show that, for the case of $k \geqslant 2$ iterations, our formula yields an over-approximation of $\bigcup_{k\geqslant 2} \tau^k(X)$.

$\boldsymbol{x}' \in \bigcup_{k\geqslant 2} \tau^k(X) \Longleftrightarrow \boldsymbol{x}' \in \tau\left(\bigcup_{k\geqslant 0} \tau^k(\tau(X))\right)$

$\Longleftrightarrow \exists k \geqslant 0, \exists \boldsymbol{x}_0 \in \tau(X), \exists \boldsymbol{x}_k : \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \wedge \boldsymbol{x}_k = \boldsymbol{x}_0 + k\mathbf{C}\boldsymbol{d} \wedge G(\boldsymbol{x}_0) \ \wedge \ \forall k' \in [1,k] \ : \ G(\boldsymbol{x}_0 + k'\mathbf{C}\boldsymbol{d})$

$\Longleftrightarrow \exists k \geqslant 0, \exists \boldsymbol{x}_0 \in \tau(X), \exists \boldsymbol{x}_k : \ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \ \wedge \ \boldsymbol{x}_k = \boldsymbol{x_0} + k\mathbf{C}\boldsymbol{d} \ \wedge \ G(\boldsymbol{x}_0) \ \wedge \ G(\boldsymbol{x}_k)$

(because G is convex)

$\Longrightarrow \exists \alpha \geqslant 0, \exists \boldsymbol{x}_0 \in \tau(X), \exists \boldsymbol{x}_k : \ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \ \wedge \ \boldsymbol{x}_k = \boldsymbol{x}_0 + \alpha\mathbf{C}\boldsymbol{d} \ \wedge \ G(\boldsymbol{x}_0)$

(dense approximation; $G(\boldsymbol{x}_k)$ implied by $\boldsymbol{x}' \in \tau(\boldsymbol{x}_k)$)

$\Longleftrightarrow \exists \boldsymbol{x}_0 \in \tau(X) \sqcap G, \exists \boldsymbol{x}_k \ : \ \boldsymbol{x}' \in \tau(\boldsymbol{x}_k) \ \wedge \ \boldsymbol{x}_k \in (\{\boldsymbol{x}_0\} \nearrow \{\mathbf{C}\boldsymbol{d}\})$

$\Longleftrightarrow \boldsymbol{x}' \in \tau\big((\tau(X) \sqcap G) \nearrow \{\mathbf{C}\boldsymbol{d}\}\big)$ □

Example 5 (Translations/resets). *(see Fig. 4) Let us consider* $\tau : x_1 + x_2 \leqslant 4 \rightarrow \left\{ \begin{array}{l} x_1' = x_1 + 2 \\ x_2' = 1 \end{array} \right.$.

Starting from $X = (0 \leqslant x_1 \leqslant 3 \ \wedge \ 2 \leqslant x_2 \leqslant 3)$ *we compute* $\tau^{\otimes}(X)$*:*

$$\begin{array}{rcl} \tau(X) & = & (2 \leqslant x_1 \leqslant 4 \ \wedge \ x_2 = 1) \\ \tau(X) \sqcap G & = & (2 \leqslant x_1 \leqslant 3 \ \wedge \ x_2 = 1) \\ \mathbf{C}\boldsymbol{d} & = & (2, 0)^T \\ (\tau(X) \sqcap G) \nearrow \{\mathbf{C}\boldsymbol{d}\} & = & (x_1 \geqslant 2 \ \wedge \ x_2 = 1) \\ \tau((\tau(X) \sqcap G) \nearrow \{\mathbf{C}\boldsymbol{d}\}) & = & (2 \leqslant x_1 \leqslant 5 \ \wedge \ x_2 = 1) \\ \tau^{\otimes}(X) & = & (x_1 \geqslant 0 \ \wedge \ 1 \leqslant x_2 \leqslant 3 \ \wedge \ x_1 + 2x_2 \geqslant 4 \ \wedge \ x_1 + x_2 \leqslant 6) \end{array}$$

4.3. General Case of Finite Monoid Transitions

Let τ be a transition $\mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b} \wedge \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$ such that the powers of $\mathbf{C}$ form a finite monoid (cf. Def. 4) with $\exists p > 0, \exists l > 0 : \mathbf{C}^{p+l} = \mathbf{C}^p$, *i.e.*, the powers of $\mathbf{C}$ generate an ultimately periodic sequence with prefix p and period l. Gonnord [26] uses the periodicity condition $\exists q > 0 : \mathbf{C}^{2q} = \mathbf{C}^q$. This condition is equivalent to the one above with $q = lcm(p,l)$.

With the latter condition, τ^* can be rewritten by enumerating the transitions induced by the powers of $\mathbf{C}$:

$$\tau^*(X) = \bigcup_{0 \leqslant j \leqslant q-1} (\tau^q)^*(\tau^j(X)) \tag{1}$$

This means that one only has to know how to accelerate τ^q which equals:

$$\tau^q = \underbrace{\bigwedge_{0 \leqslant i \leqslant q-1} \left(\mathbf{A}\mathbf{C}^i\boldsymbol{x} + \sum_{0 \leqslant j \leqslant i-1} \mathbf{C}^j\boldsymbol{d} \leqslant \boldsymbol{b} \right)}_{\mathbf{A}'\boldsymbol{x} \leqslant \boldsymbol{b}'} \rightarrow \underbrace{\boldsymbol{x}' = \mathbf{C}^q\boldsymbol{x} + \sum_{0 \leqslant j \leqslant q-1} \mathbf{C}^j\boldsymbol{d}}_{\boldsymbol{x}' = \mathbf{C}'\boldsymbol{x} + \boldsymbol{d}'} \tag{2}$$

The periodicity condition above implies that $\mathbf{C}^q$ is diagonizable and all eigenvalues of $\mathbf{C}^q$ are in $\{0,1\}$ as stated by the following lemma. Let denote $\mathbf{C}' = \mathbf{C}^q$.

Lemma 4 (Diagonizable with eigenvalues in $\{0,1\}$). $\mathbf{C}'$ *is diagonizable and all its eigenvalues are in* $\{0,1\}$ *iff* $\mathbf{C}' = \mathbf{C}'^2$.

PROOF:

($\Longrightarrow$): For any diagonal matrix $\mathbf{C}' = diag(c_1,\ldots,c_n)$ with $c_i \in \{0,1\}$ we trivially have $\mathbf{C}' = \mathbf{C}'^2$.

($\Longleftarrow$): $\mathbf{C}' = \mathbf{C}'^2$ implies that $\mathbf{C}'$ and $\mathbf{C}'^2$ have the same set of eigenvalues $\lambda_1,\ldots,\lambda_n$. Since $diag(\lambda_1,\ldots,\lambda_n)^2 = diag(\lambda_1^2,\ldots,\lambda_n^2)$, each λ_i, $i \in [1,n]$ must satisfy $\lambda_i = \lambda_i^2$ with the only solutions $\lambda_i \in \{0,1\}$. □

Hence, τ^q is a *translation with resets in the eigenbasis of* $\mathbf{C}^q$.

Lemma 5 (Translation with resets in the eigenbasis). *A transition* $\tau : \mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b} \rightarrow \boldsymbol{x}' = \mathbf{C}'\boldsymbol{x} + \boldsymbol{d}$ *where* $\mathbf{C}' = \mathbf{C}'^2$ *is a translation with resets* τ' *in the eigenbasis of* $\mathbf{C}'$*:*

$$\tau' : \mathbf{A}\mathbf{Q}\boldsymbol{x} \leqslant \boldsymbol{b} \rightarrow \boldsymbol{x}' = \mathbf{Q}^{-1}\mathbf{C}'\mathbf{Q}\boldsymbol{x} + \mathbf{Q}^{-1}\boldsymbol{d}$$

where $\mathbf{Q}^{-1}\mathbf{C}'\mathbf{Q} = diag(\lambda_1,\ldots,\lambda_n)$ *and* λ_i *the eigenvalues of* $\mathbf{C}'$*.*

By putting together, these results we get the theorem for abstract acceleration of finite monoid transitions:

Theorem 7 (Finite monoid). *Let* τ *be a transition* $G \wedge \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$ *where* $\exists q > 0 : \mathbf{C}^{2q} = \mathbf{C}^q$*, then for every convex polyhedron* X*, the convex polyhedron*

$$\tau^{\otimes}(X) = \bigsqcup_{0 \leqslant j \leqslant q-1} (\tau^q)^{\otimes}(\tau^j(X))$$

is a convex over-approximation of $\tau^*(X)$*, where* τ^q *is defined by Eq. 2 and* $(\tau^q)^{\otimes}$ *is computed using Lem. 5.*

Example 6 (Finite monoid). *(see Fig. 5)* $\tau : x_1 + x_2 \leqslant 6 \wedge \begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 2 \\ 1 \end{pmatrix}$
We have $\mathbf{C}^2 = \mathbf{I} = \mathbf{C}^4$*, thus* $q = 2$*. Obviously* $\mathbf{C}^2$ *has its eigenvalues in* $\{0,1\}$ *and* $\mathbf{Q} = \mathbf{I}$*. According to Eq. 2 we strengthen the guard by* $\mathbf{A}(\mathbf{C}\boldsymbol{x} + \boldsymbol{d}) \leqslant \boldsymbol{b} = (x_1 + x_2 \leqslant 3)$ *and compute* $\boldsymbol{d}' = (\mathbf{C}+\mathbf{I})\boldsymbol{d} = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$*. Hence we get:* $\tau^2 : (x_1 + x_2 \leqslant 3) \wedge \begin{pmatrix} x_1' \\ x_2' \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 3 \\ 3 \end{pmatrix}$
Starting from $X = (x_1 = 1 \wedge x_2 = 0)$ *we compute* $\tau^{\otimes}(X) = X \sqcup (\tau^2)^{\otimes}(X) \sqcup (\tau^2)^{\otimes}(\tau(X))$*:*

$$\begin{array}{rcl} \tau(X) & = & (x_1 = x_2 = 2) \\ (\tau^2)^{\otimes}(X) & = & (1 \leqslant x_1 \leqslant \frac{11}{2} \wedge x_1 - x_2 = 1) \\ (\tau^2)^{\otimes}(\tau(X)) & = & (1 \leqslant x_1 \leqslant 6 \wedge x_1 = x_2) \\ \tau^{\otimes}(X) & = & (2x_1 - x_2 \geqslant 2 \wedge x_2 \leqslant x_1 \wedge x_1 - x_2 \leqslant 1 \wedge x_1 + x_2 \leqslant 12) \end{array}$$

4.4. Non-Finite-Monoid Cases

In the previous section, we have shown how to handle finite monoid transformations. Now, we will generalize abstract acceleration to the additional case of linearly accelerable transformations we identified in §3.1, *i.e.*, *Jordan blocks of size two* with eigenvalues that are roots of unity. We will show here the case where $\lambda = 1$ and refer to [42] for a general presentation.

Proposition 1 (A non-finite-monoid case). *Let* $\tau : \mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b} \rightarrow \boldsymbol{x}' = \mathbf{C}\boldsymbol{x}$ *be a transition with* $\mathbf{C} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$. *Then* $\tau^{\otimes}(X) = X \sqcup \tau\big((X \sqcap G) \nearrow D\big)$ *with* $D = \big(\exists x_1 : (X \sqcap G)\big)$ *is a sound over-approximation of* $\tau^*(X)$.

The polyhedron D originates from the right upper entry in $\mathbf{C}^k\boldsymbol{x} = \begin{pmatrix} x_1 & kx_2 \\ 0 & x_2 \end{pmatrix}$ where it over-approximates the set $\{x_2 \mid \boldsymbol{x} \in X \sqcap G\}$ by which the first dimension is translated. The projection on the non-translated dimension (x_2) may incur an over-approximation if the non-translated dimension is not independent from the translated dimension (x_1) in the initial set X. Remark that we do *not* require as in Thm. 3 that the variable corresponding to the non-translated dimension has a finite number of values in the initial set: this generalization is justified by the dense approximation that we perform. For the proof, see [42]. Fig. 2 in §3.2 depicts the result of the application of Prop. 1 to Ex. 3.

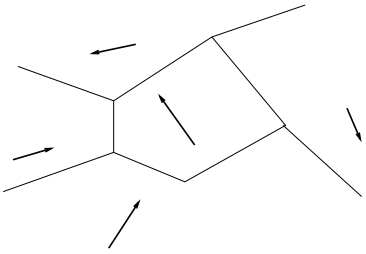
The methods proposed in this section allow us to accelerate single self-loops. However, real programs often have locations with multiple self-loops, which we will consider in the next section.

5. Abstract Acceleration of Multiple loops

In the case of flat systems (each location has at most one self-loop), we were able to replace accelerable self-loops τ by accelerated transitions $\tau^{\otimes}$. If all self-loops are accelerable, widening is not needed at all. In non-flat systems, *i.e.*, systems with nested loops, however, this is no more possible for all loops: only single inner self-loops can be replaced by their accelerated versions, and widening is needed to handle outer loops.

In this section, we propose abstract accelerations to avoid widening in cases of locations with multiple self-loops.

Remark 1. *Piecewise constant derivative (PCD) systems [43] can be considered continuous versions of multiple self-loops with translations, with the restriction that the guards are assumed to divide the space into polyhedral regions (points on borders belong to all adjacent regions). Such a system (in dimension 2) is drawn on the right-hand side. Each region is associated with a translation vector that determines the direction of the trajectory emanating of a reachable point in the region. The reachability problem for PCDs in dimension 3 and higher is undecidable.*



For the two loops, we have to compute: $\tau_1(X) \sqcup \tau_2(X) \sqcup \tau_2 \circ \tau_1(X) \sqcup \tau_1 \circ \tau_2(X) \sqcup \tau_1^2(X) \sqcup \tau_2^2(X) \sqcup \tau_1 \circ \tau_2 \circ \tau_1(X) \sqcup \tau_2 \circ \tau_1 \circ \tau_2(X) \sqcup \ldots$ We can speed up the computation of the limit of this sequence by replacing $\tau_i(X)$ by $\tau_i^{\otimes}(X)$ (because $\tau_i(X) \in \tau_i^{\otimes}(X)$), and compute the following fixed point using Kleene iteration:

$$(\bigcup_{1 \leqslant i \leqslant N} \tau_i)^{\otimes}(X_0) = lfp\lambda X.X_0 \sqcup \bigsqcup_{1 \leqslant i \leqslant N} \tau_i^{\otimes}(X) \tag{3}$$

However, this fixed point computation may not converge, thus in general, widening is necessary to guarantee termination. In our experiments, we observed that the fixed point is often reached after a few iterations in practice.

Graph expansion. The first idea [26, 27] to deal with multiple self-loops $\tau_1, \ldots, \tau_N$ was to partition the loop guards into the overlapping regions (pairwise $G_i \sqcap G_j$) and expand the graph such that each location has only a single self-loop accelerating the transitions in one of these regions. However, this approach suffers from the combinatorial explosion in the number of regions, *i.e.*, locations, and it introduces many new widening points in the resulting strongly connected graph. Hence, we will investigate cases where we are able to compute abstract accelerations of multiple loops without the use of widening.

5.1. Multiple Translation Loops

We first consider two translation loops τ_1 and τ_2: We observe that if one of the transitions is never taken ($\tau_1^{\otimes}(X) \sqcap G_2 = \bot$ or $\tau_2^{\otimes}(X) \sqcap G_1 = \bot$ respectively), then we can resort to the case of single self-loops. If both transitions may be taken, then we can further distinguish the frequent case where the guards of the transitions can be mutually strengthened, *i.e.*, substituting $G_1 \wedge G_2$ for the guards in both transitions has no influence on the concrete semantics. This gives rise to the following proposition:

Proposition 2 (Translation loops with strengthened guards). *Let* $\tau_1 : G_1 \to \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d}_1$, $\tau_2 : G_2 \to \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d}_2$, *with* $X \subseteq G_1 \sqcap G_2$ *(*) and*

$$(G_1 + \{\boldsymbol{d}_1\}) \not\sqsubseteq G_1 \ \wedge\ (G_2 + \{\boldsymbol{d}_1\}) \sqsubseteq G_2 \ \wedge\ (G_2 + \{\boldsymbol{d}_2\}) \not\sqsubseteq G_2 \ \wedge\ (G_1 + \{\boldsymbol{d}_2\}) \sqsubseteq G_1 \qquad (**)$$

then

$$\tau_{1,2}^{\otimes}(X) = X \sqcup \Big(\big((X \nearrow D) \sqcap G_1 \sqcap G_2 \big) + D \Big)$$

with $D = \{\boldsymbol{d}_1\} \sqcup \{\boldsymbol{d}_2\}$ *is a sound over-approximation of* $(\tau_1 \cup \tau_2)^*(X)$.

Intuitively, combining the two translation vectors $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$ by their convex hull D results in a time elapse that contains all interleavings of translations $\boldsymbol{d}_1$ and $\boldsymbol{d}_2$. Condition (**) states that no application of τ_1 to a state in the intersection of the guards can violate the guard of τ_2, hence strengthening, *i.e.*, conjoining, the guard of τ_2 to G_1 has no influence on τ_1, and vice versa for τ_2.

PROOF:

$$\begin{array}{ll}
 & \boldsymbol{x}' \in (\tau_1 \cup \tau_2)^*(X) \\
\iff & \exists K \geqslant 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0 \in X : \big(G_1(\boldsymbol{x}_0) \vee G_2(\boldsymbol{x}_0)\big), \forall k \in [1,K] : \exists \boldsymbol{x}_k : \\
 & \big(\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \boldsymbol{d}_1 \wedge G_1(\boldsymbol{x}_{k-1}) \ \vee\ \boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \boldsymbol{d}_2 \wedge G_2(\boldsymbol{x}_{k-1})\big) \ \wedge \\
 & \big(\boldsymbol{x}' = \tau_1(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \tau_2(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \boldsymbol{x}\big) \\
\iff & \exists K \geqslant 0, \exists \boldsymbol{x}_0 \in (X \sqcap G_1 \sqcap G_2), \forall k \in [1,K] : \exists \boldsymbol{x}_k : \qquad \text{(because of condition (*))} \\
 & \big(\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \boldsymbol{d}_1 \wedge G_1(\boldsymbol{x}_{k-1}) \wedge G_2(\boldsymbol{x}_{k-1}) \ \vee\ \boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \boldsymbol{d}_2 \wedge G_2(\boldsymbol{x}_{k-1}) \wedge G_1(\boldsymbol{x}_{k-1})\big) \ \wedge \\
 & \big(\boldsymbol{x}' = \tau_1(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \tau_2(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \boldsymbol{x}_0\big) \qquad \text{(because of condition (**))} \\
\implies & \exists \alpha \geqslant 0, \exists \boldsymbol{x}_0 \in X, \exists \boldsymbol{d} \in \{\boldsymbol{d}_1\} \sqcup \{\boldsymbol{d}_2\}, \exists \boldsymbol{x}_K \in (G_1 \sqcap G_2) : \\
 & \boldsymbol{x}_K = \boldsymbol{x}_0 + \alpha \boldsymbol{d} \ \wedge\ \big(\boldsymbol{x}' = \boldsymbol{x}_K + \boldsymbol{d} \ \vee\ \boldsymbol{x}' = \boldsymbol{x}_0\big) \qquad \text{(convex and dense approximation)} \\
\iff & X \sqcup \Big(\big((X \nearrow D) \sqcap G_1 \sqcap G_2 \big) + D \Big)
\end{array}$$

□

Example 7 (Translation loops with strengthened guards). *(see Fig. 6)*
$\tau_1 : x_1+x_2 \leqslant 4 \to \begin{cases} x_1' = x_1+2 \\ x_2' = x_2+1 \end{cases}$ *and* $\tau_2 : -2x_1+x_2 \leqslant 4 \to \begin{cases} x_1' = x_1-1 \\ x_2' = x_2+1 \end{cases}$.
Starting from $X = (0 \leqslant x_1 \leqslant 1 \wedge 0 \leqslant x_2 \leqslant 1)$*, we compute:*

$$\begin{array}{rcl} D & = & (-1 \leqslant x_1 \leqslant 2 \ \wedge \ x_2 = 1) \\ X \nearrow D & = & (x_2 \geqslant -x_1 \ \wedge \ x_2 \geqslant 0 \ \wedge \ x_1 - 2x_2 \leqslant 1) \\ (X \nearrow D) \sqcap G_1 \sqcap G_2 & = & (x_2 \geqslant -x_1 \ \wedge \ x_2 \geqslant 0 \ \wedge \ x_1 - 2x_2 \leqslant 1 \ \wedge \ x_1 + x_2 \leqslant 4 \ \wedge \ -2x_1 + x_2 \leqslant 4) \\ (\tau_1 \cup \tau_2)^{\otimes}(X) & = & (x_2 \geqslant -x_1 \ \wedge \ 0 \leqslant x_2 \leqslant 5 \ \wedge \ x_1 - 2x_2 \leqslant 1 \ \wedge \ x_1 + x_2 \leqslant 7 \ \wedge \ -2x_1 + x_2 \leqslant 7) \end{array}$$

The polyhedron D includes all vectors pointing upwards between $(-1,1)$ *and* $(2,1)$*, thus,* $X \nearrow D$ *corresponds to the unbounded polyhedron above the three lower line segments (including the dashed lines) in Fig. 6. The final result is then obtained by intersecting with both guards and joining the results obtained from applying* τ_1 *and* τ_2*, respectively.*

The descending iterations in standard LRA cannot recover the three upper-bounding constraints, and therefore it yields $x_2 \geqslant -x_1 \ \wedge \ x_2 \geqslant 0 \ \wedge \ x_1 - 2x_2 \leqslant 1$.

In the general case of two translation loops, we can safely approximate $(\tau_1 \cup \tau_2)^*(X)$ using the following proposition:

Proposition 3 (Translation loops). *Let* $\tau_1 : G_1 \to \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d}_1$ *and* $\tau_2 : G_2 \to \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d}_2$*, then*

$$\tau_{1,2}^{\otimes}(X) = X \sqcup \tau_1(Y) \sqcup \tau_2(Y) \quad \textit{with} \quad Y = \big((X \sqcap G_1) \sqcup (X \sqcap G_2)\big) \nearrow \big(\{\boldsymbol{d}_1\} \sqcup \{\boldsymbol{d}_2\}\big)$$

is a sound over-approximation of $(\tau_1 \cup \tau_2)^*(X)$.

In contrast to Prop. 2, we cannot exploit information about the guards in the last iteration. The join $\tau_1(Y) \sqcup \tau_2(Y)$ computes just a normal descending iteration and hence, if applied to Ex. 7, it yields the same result as standard LRA.

More than two translation loops. Prop. 3 generalizes to N translation loops as follows:

$$\Big(\bigcup_{1 \leqslant i \leqslant N} \tau_i\Big)^{\otimes}(X) = X \sqcup \bigsqcup_{1 \leqslant i \leqslant N} \tau_i\Big(\Big(\bigsqcup_{1 \leqslant i \leqslant N} X \sqcap G_i\Big) \nearrow \Big(\bigsqcup_{1 \leqslant i \leqslant N} \{\boldsymbol{d}_i\}\Big)\Big)$$

Similarly, Prop. 2 can be generalized to

$$\tau_{1\ldots N}^{\otimes}(X) = X \sqcup \Big(\big((X \nearrow D) \sqcap \prod_{1 \leqslant i \leqslant N} G_i\big) + D\Big) \quad \text{with} \quad D = \Big(\bigsqcup_{1 \leqslant i \leqslant N} \{\boldsymbol{d}_i\}\Big).$$

In practice, one starts computing the Kleene iteration with the individually accelerated transitions $\tau_i^{\otimes}(X)$. Sometimes a fixed point is already reached after a few iterations. Then one can apply Prop. 2 if its preconditions are fulfilled or otherwise resort to Prop. 3. Gonnord and Halbwachs describe some other heuristics in [27].

5.2. Multiple Translation and Reset Loops

We start with the simplest case: a translation loop and a reset loop.

Proposition 4 (Translation and reset loop). *Let* $\tau_1 : G_1 \to \boldsymbol{x}' = \boldsymbol{x}+\boldsymbol{d}_1$ *and* $\tau_2 : G_2 \to \boldsymbol{x}' = \boldsymbol{d}_2$*, then*

$$\tau_{1,2}^{\otimes}(X) = \tau_1^{\otimes}(X \sqcup \tau_2(\tau_1^{\otimes}(X)))$$

is a sound over-approximation of $(\tau_1 \cup \tau_2)^*(X)$.

Intuitively, the translation τ_1 can only act on the initial set X and the reset state $\boldsymbol{d}_2$ (if τ_2 can be taken at all, *i.e.*, G_2 must be satisfied by X or any iteration of τ_1 on X).

PROOF:

$$\begin{array}{rcll}
 & (\tau_1 \cup \tau_2)^* & & \\
= & id \cup \tau_1 \cup \tau_2 \cup \tau_2 \circ \tau_1 \cup \tau_1 \circ \tau_2 \cup \tau_1^2 \cup \tau_2^2 \cup \tau_2 \circ \tau_1^2 \cup \tau_2^2 \circ \tau_1 \cup \dots & & \\
= & id \cup \tau_1 \cup \tau_2 \cup \tau_2 \circ \tau_1 \cup \tau_1 \circ \tau_2 \cup \tau_1^2 \cup \tau_1^2 \circ \tau_2 \cup \dots & & (\tau_2 \text{ is idempotent}) \\
\subseteq & \tau_1^* \cup \tau_2 \circ \tau_1^* \cup \tau_1^* \circ \tau_2 \cup \tau_1^* \circ \tau_2 \circ \tau_1^* \cup \tau_2 \circ \tau_1^* \circ \tau_2 \cup \dots & & (\tau_1^* \text{ is idempotent, contains } id \text{ and } \tau_1) \\
= & \tau_1^* \cup \tau_2 \circ \tau_1^* \cup \tau_1^* \circ \tau_2 \cup \tau_1^* \circ \tau_2 \circ \tau_1^* & & (\tau_2 \text{ and } \tau_1^* \circ \tau_2 \text{ are idempotent}) \\
= & \tau_1^* \circ (id \cup (\tau_2 \circ \tau_1^*)) & &
\end{array}$$

By the monotonicity of the abstraction, we have $(\alpha \circ \tau_1^* \circ (id \cup (\tau_2 \circ \tau_1^*) \circ \gamma))(X) \sqsubseteq \tau_1^{\otimes}(X \sqcup \tau_2(\tau_1^{\otimes}(X)))$. □

Example 8 (Translation and reset loop). *(see Fig. 7)* $\tau_1 : x_1+x_2 \leqslant 4 \to \left\{ \begin{array}{l} x_1' = x_1+1 \\ x_2' = x_2+1 \end{array} \right.$ *and* $\tau_2 : x_1+x_2 \geqslant 5 \to \left\{ \begin{array}{l} x_1' = 2 \\ x_2' = 0 \end{array} \right.$. *Starting from* $X = (x_1 = x_2 = 0)$*, we compute:*

$$\begin{array}{rcl}
\tau_1^{\otimes}(X) & = & (0 \leqslant x_1 = x_2 \leqslant 3) \\
\tau_2\left(\tau_1^{\otimes}(X)\right) & = & (x_1 = 2 \wedge x_2 = 0) \\
\tau_2\left(\tau_1^{\otimes}(X)\right) \sqcup X & = & (0 \leqslant x_1 \leqslant 2 \wedge x_2 = 0) \\
\tau_1^{\otimes}\left(\tau_2\left(\tau_1^{\otimes}(X)\right) \sqcup X\right) & = & (0 \leqslant x_2 \leqslant x_1 \leqslant x_2+2 \wedge x_1+x_2 \leqslant 6)
\end{array}$$

$\tau_1^{\otimes}(X)$ gives us the line segment from $(0,0)$ to $(3,3)$. The application of τ_2 adds to the result the second "start" point $(2,0)$ that is joined with X, and re-applying $\tau_1^{\otimes}$ gives us the final result. Since τ_2 is monotonic, we only need to apply it to the bigger set $\tau_1^{\otimes}(X)$ instead of X.

Similar to Prop. 2, we can consider the case where the guards of the transitions can be mutually strengthened:

Proposition 5 (Translation and translation/reset loops with strengthened guards). *Let* $\tau_1 : G_1 \to \boldsymbol{x}' = \boldsymbol{x}+\boldsymbol{d}_1$*,* $\tau_2 : G_2 \to \boldsymbol{x}' = \mathbf{C}_2\boldsymbol{x}+\boldsymbol{d}_2$ *with*

$\big(X \sqcup \tau_2(X)\big) \subseteq (G_1 \sqcap G_2)$ *and* *(*)*

$(G_1+\{\boldsymbol{d}_1\}) \not\sqsubseteq G_1 \wedge (G_2+\{\boldsymbol{d}_1\}) \sqsubseteq G_2 \wedge (G_2+\{\mathbf{C}_2\boldsymbol{d}_2\}) \not\sqsubseteq G_2 \wedge (G_1+\{\mathbf{C}_2\boldsymbol{d}_2\}) \sqsubseteq G_1$ *(**)*

then

$$\tau_{1,2}^{\otimes}(X) = X \sqcup \Big(\big(((X \sqcup \tau_2(X)) \nearrow D) \sqcap G_1 \sqcap G_2\big) + D\Big)$$

with $D = \big(\bigsqcup\big\{\{\boldsymbol{d}_1\}, \{\mathbf{C}_2\boldsymbol{d}_2\}, \{\mathbf{C}_2\boldsymbol{d}_1\}\big\}\big)$ *is a sound over-approximation of* $(\tau_1 \cup \tau_2)^*(X)$.

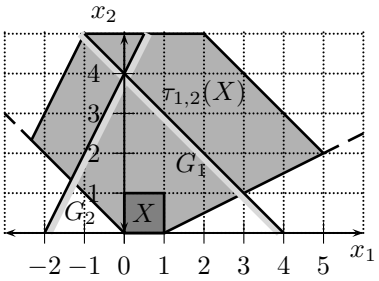


Figure 6: Acceleration for translation loops with strengthened guards (Ex. 7).

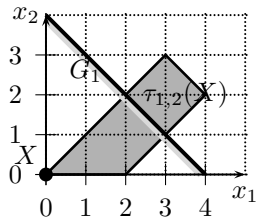


Figure 7: Abstract acceleration of translation and reset loops (Ex. 8).

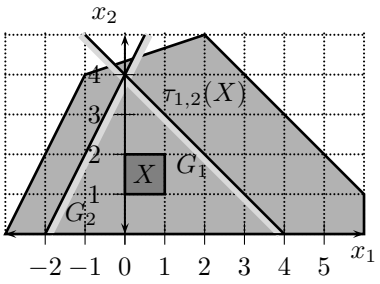


Figure 8: Acceleration for translation and translation/reset loops with strengthened guards (Ex. 9).

There are two differences to Prop. 2: firstly, $\tau_2(X)$ is applied to expand the initial set with the reset values of τ_2 (cf. Thm. 6), and secondly, D contains, besides the translation vectors of τ_1 and τ_2, the projection of $\boldsymbol{d}_1$ on the translated dimensions of τ_2. The explanation for the latter is that the alternating application of τ_1 and τ_2 produces sawtooth-like trajectories of which the direction of the baseline is given by $\mathbf{C}_2\boldsymbol{d}_1$.[4]

PROOF: Notation: we partition the dimensions of the vectors $\boldsymbol{x}$ according to $\mathbf{C}_2 = diag(c_1 \dots c_n)$ into $\boldsymbol{x}^t$ consisting of the translated dimensions ($c_i = 1$), and the reset dimensions $\boldsymbol{x}^r$ ($c_i = 0$).

$$
\begin{aligned}
& \boldsymbol{x}' \in (\tau_1 \cup \tau_2)^*(X) \\
\iff\ & \exists K \geqslant 0, \exists \boldsymbol{x} \in X, \exists \boldsymbol{x}_0 \in X : \big(G_1(\boldsymbol{x}_0) \vee G_2(\boldsymbol{x}_0)\big) : \forall k \in [1,K] : \exists \boldsymbol{x}_k : \\
& \big(\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \boldsymbol{d}_1 \wedge G_1(\boldsymbol{x}_{k-1}) \ \vee\ \boldsymbol{x}_k = \mathbf{C}_2\boldsymbol{x}_{k-1} + \boldsymbol{d}_2 \wedge G_2(\boldsymbol{x}_{k-1})\big) \ \wedge \\
& \big(\boldsymbol{x}' = \tau_1(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \tau_2(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \boldsymbol{x}\big) \\
\iff\ & \exists K \geqslant 0, \exists \boldsymbol{x}_0 \in \big((X \sqcup \tau_2(X)) \sqcap G_1 \sqcap G_2\big) : \forall k \in [1,K] : \exists \boldsymbol{x}_k : \qquad \text{(because of (*))} \\
& \big(\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \boldsymbol{d}_1 \wedge G_1(\boldsymbol{x}_{k-1}) \wedge G_2(\boldsymbol{x}_{k-1}) \ \vee\ \boldsymbol{x}_k = \mathbf{C}_2\boldsymbol{x}_{k-1} + \boldsymbol{d}_2 \wedge G_1(\boldsymbol{x}_{k-1}) \wedge G_2(\boldsymbol{x}_{k-1})\big) \ \wedge \\
& \big(\boldsymbol{x}' = \tau_1(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \tau_2(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \boldsymbol{x}\big) \qquad \text{(because of (**))} \\
\implies\ & \exists K \geqslant 0, \exists \boldsymbol{x}_0 \in \big((X \sqcup \tau_2(X)) \sqcap G_1 \sqcap G_2\big), \exists \boldsymbol{x}_K : \\
& \big(\boldsymbol{x}^t_K = \boldsymbol{x}^t_0 + K\boldsymbol{d}^t_1 \ \vee\ \boldsymbol{x}^t_K = \boldsymbol{x}^t_0 + K\boldsymbol{d}^t_2\big) \wedge \big(\boldsymbol{x}^r_K = \boldsymbol{x}^r_0 + K\boldsymbol{d}^r_1 \ \vee\ \boldsymbol{x}^r_K = \boldsymbol{d}^r_2 \ \vee\ \boldsymbol{x}^r_K = \boldsymbol{d}^r_2 + K\boldsymbol{d}^r_1\big) \wedge \\
& \big(\boldsymbol{x}' = \tau_1(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \tau_2(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \boldsymbol{x}\big) \\
\implies\ & \exists \alpha \geqslant 0, \exists \boldsymbol{x}_0 \in \big((X \sqcup \tau_2(X)) \sqcap G_1 \sqcap G_2\big), \exists \boldsymbol{d} \in \{\boldsymbol{d}_1\} \sqcup \{\mathbf{C}_2\boldsymbol{d}_2\} \sqcup \{\mathbf{C}_2\boldsymbol{d}_1\}, \exists \boldsymbol{x}_K : \\
& \boldsymbol{x}_K = \boldsymbol{x}_0 + \alpha\boldsymbol{d} \ \wedge\ \big(\boldsymbol{x}' = \tau_1(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \tau_2(\boldsymbol{x}_K) \ \vee\ \boldsymbol{x}' = \boldsymbol{x}\big) \\
& \quad (\boldsymbol{d} \text{ is in the convex hull of } \{\boldsymbol{d}^t_1, \boldsymbol{d}^t_2\} \times \{\boldsymbol{d}^r_1, \mathbf{0}^r\}, \text{ and } \{\boldsymbol{x}^t_0\} \times \{\boldsymbol{x}^r_0, \boldsymbol{d}^r_2\} \subseteq X \sqcup \tau_2(X)\ (*)) \\
\iff\ & \boldsymbol{x}' \in X \sqcup \Big(\big((X \nearrow D) \sqcap G_1 \sqcap G_2\big) + D\Big) \ \wedge\ D = \big(\bigsqcup \big\{\{\boldsymbol{d}_1\}, \{\mathbf{C}_2\boldsymbol{d}_2\}, \{\mathbf{C}_2\boldsymbol{d}_1\}\big\}\big) \qquad \square
\end{aligned}
$$

Example 9 (Translation and translation/reset loops). *(see Fig. 8)*

$$\tau_1 : x_1 + x_2 \leqslant 4 \to \begin{cases} x'_1 = x_1 + 2 \\ x'_2 = x_2 + 1 \end{cases} \text{ and } \tau_2 : -2x_1 + x_2 \leqslant 4 \to \begin{cases} x'_1 = x_1 - 1 \\ x'_2 = 0 \end{cases}.$$

[4] A more precise formula can be devised by considering $\mathbf{C}_2(\boldsymbol{d}_1 + \boldsymbol{d}_2)$, and applying τ_1 and τ_2 instead of adding D after guard intersection. We leave the proof to the interested reader.

Starting from $X = (0 \leqslant x_1 \leqslant 1 \wedge 1 \leqslant x_2 \leqslant 2)$*, we compute:*

$$\begin{array}{rcl} \tau_2(X) & = & (x_2 = 0 \ \wedge \ x_1 \leqslant 0 \ \wedge \ -1 \leqslant x_1) \\ X \sqcup \tau_2(X) & = & (x_1 \leqslant x_2 \ \wedge \ 0 \leqslant x_2 \ \wedge \ x_1 \leqslant 1 \ \wedge \ x_2 \leqslant 2 \ \wedge \ x_2 \leqslant 2x_1 + 2) \\ D & = & (2,1) \sqcup (-1,0) \sqcup (2,0) = (3x_2 \leqslant x_1 + 1 \ \wedge \ 0 \leqslant x_2 \ \wedge \ x_1 \leqslant 2) \\ (X \sqcup \tau_2(X)) \nearrow D & = & (x_2 \geqslant 0) \\ \tau_{1,2}^{\otimes}(X) & = & (x_2 \geqslant 0 \ \wedge \ x_1 \leqslant 6 \ \wedge \ x_1 + x_2 \leqslant 7 \ \wedge \ -2x_1 + x_2 \leqslant 6 \ \wedge \ -x_1 + 3x_2 \leqslant 13) \end{array}$$

In [18], Gonnord and Halbwachs proposed a specialized abstract acceleration for another combination of translation and translation/reset loops which frequently occurs in synchronous controller programs.

Proposition 6 ([18]). *Let* $\tau_1 : G_1 \to \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d}_1$ *and* $\tau_2 : \boldsymbol{x}' = \mathbf{C}_2\boldsymbol{x} + \boldsymbol{d}_2$ *with* $G_1 = (x_n \leqslant K)$*,* $\mathbf{C}_2 = diag(1 \ldots 1, 0)$ *and*[5] $d_{2,n} = 0$*, and assume* $X \sqsubseteq (x_n = K)$*. Then*

$$\tau_{1,2}^{\otimes}(X) = \begin{cases} X \nearrow \bigsqcup\{\boldsymbol{d}_1, \mathbf{C}_2\boldsymbol{d}_1, \boldsymbol{d}_2\} & \text{if } X \nearrow \{\boldsymbol{d}_1\} \sqsubseteq G_1 \\ X \sqcup \tau_1\big(X \nearrow \bigsqcup\{\boldsymbol{d}_1, \boldsymbol{d}_2, k_{max}\mathbf{C}_2\boldsymbol{d}_1 + \boldsymbol{d}_2\}\big) & \text{else} \end{cases}$$

with $k_{max} = \lfloor K/d_{1,n} \rfloor + 1$ *is a convex overapproximation of* $(\tau_1 \cup \tau_2)^*(X)$*.*

Prop. 6 is illustrated in Fig 10 for $n = 3$.

Example 10. *We consider the famous cruise control example [5] (see Fig. 9). There are two input signals that trigger a transition: the "second" signal increments time t and resets the speed estimate s to 0, and the "meter" signal increments both the distance d and the speed estimate s. The "meter" signal can only occur when* $s \leqslant 3$ *because it is assumed that the speed is less than 4m/s. On the CFG, the occurences of the two signals "second" and "meter" are abstracted by non-deterministic choices.*

With notations of figure 10, we have $\boldsymbol{x} = (t, d, s)$*,* $G_1 = (s \leqslant 3)$*,* $\boldsymbol{d}_1 = (0,1,1)$*,* $\boldsymbol{d}_2 = (1,0,0)$*, and* $X = \{(0,0,0)\}$*. Moreover, with* $k_{max} = 3/1 + 1 = 4$ *and* $\mathbf{C}_2\boldsymbol{d}_1 = (0,1,0)$ *we have* $k_{max}\mathbf{C}_2\boldsymbol{d}_1 + \boldsymbol{d}_2 = (1,4,0)$*, and we obtain:*

$$\begin{array}{rcl} \tau_{1,2}^{\otimes}(X) & = & \big(\{(0,0,0)\} \nearrow \bigsqcup\{(0,1,1),(1,0,0),(1,4,0)\}\big) \sqcap (s \leqslant 3) \\ & = & (t \geqslant 0 \ \wedge \ 0 \leqslant s \leqslant 3 \ \wedge \ 0 \leqslant d \leqslant 4t + s) \end{array}$$

For the general case of a translation and a translation/reset loop, we can give a formula by combining ideas from Propositions 3, 4 and 5:

$$\begin{array}{rcl} \tau_{1,2}^{\otimes}(X) = & X \sqcup \tau_1(Y) \sqcup \tau_2(Y) & \\ \text{with} & \begin{cases} Y = \big((Z \sqcap G_1) \sqcup (Z \sqcap G_2)\big) \nearrow \big(\bigsqcup\big\{\{\boldsymbol{d}_1\}, \{\mathbf{C}_2\boldsymbol{d}_2\}, \{\mathbf{C}_2\boldsymbol{d}_1\}\big\}\big) \\ Z = \tau_2(\tau_1^{\otimes}(X)) \sqcup X \end{cases} & \end{array}$$

However, its generalization to more than two loops is less obvious.

These results led to algorithms that were implemented in the ASPIC tool (§7). More recent work considers a different approach to handling multiple loops, as we will see in the next section.

[5] $d_{i,n}$ denotes the n^{th} dimension of $\boldsymbol{d}_i$.

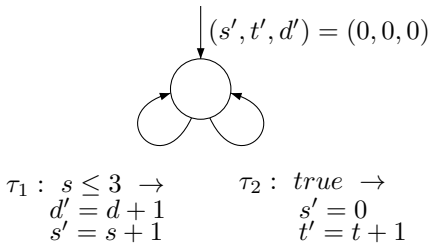


Figure 9: CFG of the cruise control Ex. 10.

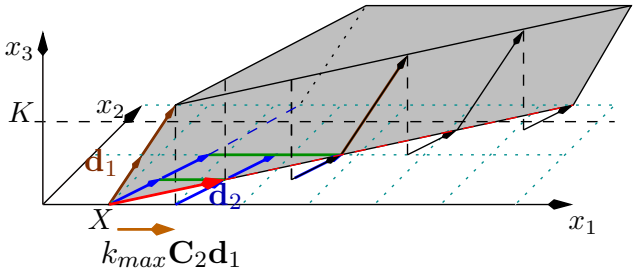


Figure 10: Abstract acceleration of translation and translation/reset loops (Prop. 6).

5.3. Path Focusing

Henry and al [44] and Monniaux and Gonnord [45] propose to run classical Kleene iterations on a modified CFG that consists only of a set of "cut" nodes (every cycle in the initial CFG passes through at least one of these nodes). Joins (convex hulls) are only taken in these locations and the loop bodies are entirely represented symbolically. This has the advantage that convex unions on joining branches within the loops are avoided and hence the precision is improved. During the analysis, one searches for loop paths that violate the current invariant by posing a query to an SMT solver: Is $\Phi_G(\boldsymbol{x}, \boldsymbol{x'}) \wedge \boldsymbol{x} \in X \wedge \boldsymbol{x'} \notin X \wedge b_i^s \wedge b_i^d \wedge \ldots$ satisfiable? where Φ_G denotes an encoding of the CFG: control points and transitions are boolean values, transition are encoded with primed values; $b_i^s \wedge b_i^d$ expresses that the desired path begins and end at a given control point, and $\boldsymbol{x} \in X \wedge \boldsymbol{x'} \notin X$ means that the desired path should add new states to the current abstract value. If there is a positive answer, the valuation gives us a new path that can possibly be accelerated, or at least iterated with widening until convergence.

Non-elementary loops. Until now our results have been exposed for simple or multiple elementary loops in a control location. We can still apply abstract acceleration techniques to non-elementary loops by composing the transitions of cycles of nested loops. However, as enumerating cycles is costly, we have to choose between all cycles those that are most likely to be accelerable. Again, this choice can be made via adequate SMT-queries as explained above.

6. Extensions

This section summarizes further developments w.r.t. abstract acceleration. First, abstract acceleration can be generalized from closed to open systems, *e.g.*, reactive systems with numerical *inputs* (§6.1, [46, 47]). Then, it can also be applied to perform a *backward* (co-reachability) analysis (§6.2, [47]). At last, §6.3 describes a method that makes abstract acceleration applicable to *logico-numerical* programs, *i.e.*, programs with Boolean and numerical variables.

6.1. Reactive Systems With Numerical Inputs

Reactive programs such as LUSTRE programs [48] interact with their environment: at each computation step, they have to take into account the values of *input* variables, which typically correspond to values acquired by sensors. Boolean input variables can be encoded in a CFG by finite

non-deterministic choices, but numerical input variables require a more specific treatment. Indeed, they induce transitions of the form $\tau : G(\boldsymbol{x},\boldsymbol{\xi}) \to \boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x},\boldsymbol{\xi})$, $\boldsymbol{x},\boldsymbol{x}' \in \mathbb{R}^n$, $\boldsymbol{\xi} \in \mathbb{R}^p$ that depend on both, numerical state variables $\boldsymbol{x}$ and numerical input variables $\boldsymbol{\xi}$. We consider transitions τ where the variables $\boldsymbol{x}$, $\boldsymbol{\xi}$ occur in general linear expressions in guards and actions:

$$\underbrace{\begin{pmatrix}\mathbf{A} & \mathbf{L}\\ \mathbf{0} & \mathbf{P}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{\xi}\end{pmatrix} \leqslant \begin{pmatrix}\boldsymbol{b}\\ \boldsymbol{q}\end{pmatrix}}_{\mathbf{A}\boldsymbol{x}+\mathbf{L}\boldsymbol{\xi}\leqslant\boldsymbol{b}\ \wedge\ \mathbf{P}\boldsymbol{\xi}\leqslant\boldsymbol{q}} \to \boldsymbol{x}' = \underbrace{\begin{pmatrix}\mathbf{C} & \mathbf{T}\end{pmatrix}\begin{pmatrix}\boldsymbol{x}\\ \boldsymbol{\xi}\end{pmatrix} + \boldsymbol{u}}_{\mathbf{C}\boldsymbol{x}+\mathbf{T}\boldsymbol{\xi}+\boldsymbol{u}} \tag{4}$$

The main challenge raised by adding inputs consists in the fact that any general affine transformation without inputs $\mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b} \to \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}$ can be expressed as a "reset with inputs" $(\mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b} \wedge \boldsymbol{\xi} = \mathbf{C}\boldsymbol{x} + \boldsymbol{d}) \to \boldsymbol{x}' = \boldsymbol{\xi}$ [47]. This means that there is no hope to get precise acceleration for such resets or translations with inputs, unless we know how to accelerate precisely general affine transformations without inputs.

Nevertheless, we can accelerate transitions with inputs if the constraints on the state variables do not depend on the inputs, *i.e.*, the guard is of the form $\mathbf{A}\boldsymbol{x} \leqslant \boldsymbol{b} \wedge \mathbf{P}\boldsymbol{\xi} \leqslant \boldsymbol{q}$, *i.e.*, when $\mathbf{L} = 0$ in Eqn. (4). We call the resulting guards *simple guards*. A *general guard* G' can be relaxed to a simple guard $G = \underbrace{(\exists \boldsymbol{\xi} : G')}_{\mathbf{A}\boldsymbol{x}\leqslant\boldsymbol{b}} \wedge \underbrace{(\exists \boldsymbol{x} : G')}_{\mathbf{P}\boldsymbol{\xi}\leqslant\boldsymbol{q}}$. This trivially results in a sound over-approximation because $G \sqsupseteq G'$.

We will now show how to abstractly accelerate a *translation with inputs and simple guards*: The first step is to reduce it to a *polyhedral translation* defined as $\tau : G \to \boldsymbol{x}' = \boldsymbol{x} + D$ with the semantics $\tau(X) = (X \sqcap G) + D$. Intuitively, the polyhedron D contains all possible values (induced by the inputs ξ) that may be added to x. The polyhedron $D = \{\boldsymbol{d} \mid \boldsymbol{d} = \boldsymbol{T}\boldsymbol{\xi} + \boldsymbol{u} \wedge \boldsymbol{P}\boldsymbol{\xi} \leqslant \boldsymbol{q}\}$ can be computed using standard polyhedra operations. Then, Thm. 5 can be generalized from ordinary translations to polyhedral translations.

Theorem 8 (Polyhedral translation [47]). *Let τ be a polyhedral translation $G \to \boldsymbol{x}' = \boldsymbol{x} + D$ with simple guards, and X, D convex polyhedra. Then, the set*

$$\tau^{\otimes}(X) = X \sqcup \tau\big((X \sqcap G) \nearrow D\big)$$

is a convex over-approximation of $\tau^*(X)$.

The case of *translations/resets with inputs* can be handled similarly to translations: first, we reduce translations/resets with inputs and simple guards polyhedral translations with resets $\tau : G \to \boldsymbol{x}' = \mathbf{C}\boldsymbol{x} + D$. However, the resulting abstract acceleration becomes more complicated than Thm. 6 because, unlike in the case of resets to constants, the variables may be assigned a different value in each iteration. We refer to [47] for further details.

The general case of *finite monoid transitions* (Thm. 7) can be extended to inputs based on Eq. (1) $\tau^*(X) = \bigcup_{0\leqslant j\leqslant q-1}(\tau^q)^*(\tau^j(X))$, where, in the case of inputs, τ^q is defined as:

$$\bigwedge_{0\leqslant i\leqslant q-1}\Big(\mathbf{A}\mathbf{C}^i\boldsymbol{x} + \mathbf{L}\mathbf{C}^i\boldsymbol{\xi}_i + \sum_{0\leqslant j\leqslant i-1}\mathbf{T}\mathbf{C}^j\boldsymbol{\xi}_j - \mathbf{C}^j\boldsymbol{u} \leqslant \boldsymbol{b}\Big)\ \wedge\ (\mathbf{P}\boldsymbol{\xi} \leqslant \boldsymbol{q}) \quad \to \quad \Big(\boldsymbol{x}' = \mathbf{C}^q\boldsymbol{x} + \sum_{0\leqslant j\leqslant q-1}\mathbf{T}\mathbf{C}^j\boldsymbol{\xi}_j - \mathbf{C}^j\boldsymbol{u}\Big)$$

Then we can accelerate each τ^q by relaxing the guard, reducing them to polyhedral translations (with resets) and applying the respective theorems in the eigenbasis of $\mathbf{C}^q$ according to Lem. 5. Mind that we need to duplicate the inputs q times.

Also, the *non-finite monoid cases* of §4.4 can be extended to inputs in a straightforward way by taking them into account (existential quantification) in the computation of $D^{(l)}$ in Prop. 1.

6.2. Backward Analysis

Abstract acceleration has been applied to forward reachability analysis in order to compute the reachable states starting from a set of initial states. Backward analysis computes the states co-reachable from the *error* states. For example, combining forward and backward analysis allows to obtain an approximation of the sets of states belonging to a path from initial to error states. The experience of verification tools, *e.g.*, [49], shows that combining forward and backward analyses results in more powerful tools.

Although the inverse of a translation is a translation, the difference is that the intersection with the guard occurs after the (inverted) translation. The case of backward translations with resets is more complicated than for the forward case, because resets are not invertible. We state here just the translation case and refer to [47] for the other cases and their respective extensions to numerical inputs.

Proposition 7 (Backward translation [47]). *Let τ be a translation $G \to \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d}$. Then the set*

$$\tau^{-\otimes}(X') = X' \sqcup \left((\tau^{-1}(X') \nearrow \{-\boldsymbol{d}\}\right) \sqcap G$$

is a convex over-approximation of $\tau^{-*}(X')$, where $\tau^{-*} = (\tau^{-1})^* = (\tau^*)^{-1}$ is the reflexive and transitive backward closure of τ.

6.3. Logico-Numerical Abstract Acceleration

The classical approach to applying abstract acceleration to logico-numerical programs like LUSTRE programs [48] relies on the enumeration of the Boolean state space. However, this technique suffers from the state space explosion. In [50], Schrammel and Jeannet proposed a method that alleviates this problem by combining an analysis in a *logico-numerical abstract domain* (that represents both Boolean and numerical states symbolically) and *state space partitioning*. This approach separates the issue of (i) accelerating transition relations that contain Boolean variables from (ii) finding a suitable CFG that enables a reasonably precise and fast reachability analysis.

Self-loops in logico-numerical programs are of the form $\tau : \begin{pmatrix} \boldsymbol{b}' \\ \boldsymbol{x}' \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}^b(\boldsymbol{b}, \boldsymbol{\beta}, \boldsymbol{x}, \boldsymbol{\xi}) \\ \boldsymbol{f}^x(\boldsymbol{b}, \boldsymbol{\beta}, \boldsymbol{x}, \boldsymbol{\xi}) \end{pmatrix}$ where $\boldsymbol{f}^x$ is a vector of expressions of the form $\bigvee_j \left(a_j(\boldsymbol{x}, \boldsymbol{\xi}) \text{ if } g_j(\boldsymbol{b}, \boldsymbol{\beta}, \boldsymbol{x}, \boldsymbol{\xi})\right)$ where a_j are arithmetic expressions, and g_j (as well as the components of $\boldsymbol{f}^b$) are arbitrary (Boolean) formulas involving Boolean state variables $\boldsymbol{b}$, Boolean input variables $\boldsymbol{\beta}$ and constraints over $\boldsymbol{x}$ and $\boldsymbol{\xi}$. The "guards" g_j are assumed to be disjoint.

We use the logico-numerical abstract domain that abstracts logico-numerical state sets $\wp(\mathbb{B}^m \times \mathbb{R}^n)$ by a cartesian product of Boolean state sets $\wp(\mathbb{B}^m)$ (represented with the help of BDDs) and convex polyhedra abstracting $\wp(\mathbb{R}^n)$.

By factorizing the guards of $\boldsymbol{f}^x$, we obtain several loops of the form $\tau : g^b(\boldsymbol{b},\boldsymbol{\beta}) \wedge g^x(\boldsymbol{x},\boldsymbol{\xi}) \rightarrow \begin{pmatrix} \boldsymbol{b}' \\ \boldsymbol{x}' \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}^b(\boldsymbol{b},\boldsymbol{\beta},\boldsymbol{x},\boldsymbol{\xi}) \\ \boldsymbol{a}^x(\boldsymbol{x},\boldsymbol{\xi}) \end{pmatrix}$. Such a loop is accelerable if g^x is a conjunction of linear inequalities and $\boldsymbol{x}' = \boldsymbol{a}^x(\boldsymbol{x},\boldsymbol{\xi})$ are accelerable actions. Then numerical and Boolean parts of the transition function τ are decoupled into

$$\tau_b : g^b \wedge g^x \rightarrow \begin{pmatrix} \boldsymbol{b}' \\ \boldsymbol{x}' \end{pmatrix} = \begin{pmatrix} \boldsymbol{f}^b \\ \boldsymbol{x} \end{pmatrix} \text{ and } \tau_x : g^b \wedge g^x \rightarrow \begin{pmatrix} \boldsymbol{b}' \\ \boldsymbol{x}' \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ \boldsymbol{a}^x \end{pmatrix}.$$

and $\tau^{\otimes}$ is approximated by $(\tau_b^{\sharp})^* \circ \tau_x^{\otimes}$ where $\tau_x^{\otimes}$ is computed using numerical abstract acceleration (which is possible because its Boolean part is the identity) and the abstract version $\tau_b^{\sharp}$ of τ_b is iterated up to convergence (which is guaranteed because the lattice of Boolean state sets is finite).

This method can be applied to accelerable loops (in the sense above) in any CFG. However, in order to alleviate the impact of decoupling Boolean and numerical transition functions on the precision, it is applied to CFGs obtained by partitioning techniques that group Boolean states that exhibit the same numerical behavior ("numerical modes") in the same locations.

7. Tools and Experimental Results

In [27, 51] Gonnord et al. showed a general method to integrate abstract acceleration results onto a classical LRA tool. Basically, a preprocessing phase identifies *accelerable loops* and precomputes the static information used for acceleration. Accelerable loops are tagged so that to avoid widening application on their heads. Then a classical LRA is performed, where the application of any accelerable loop is replaced by the application of its acceleration.

7.1. Tools for abstract acceleration

ASPIC[6] was the first implementation of abstract acceleration techniques for numerical programs. It implements a combination of classical LRA with abstract acceleration on polyhedra on accelerable simple loops and multiple loops (§4 and §5). The implementation details can be found in [51]. ASPIC takes as input a variant of the FAST [17] format, and is able to compute numerical invariants for C programs via C2FSM [51]. It also belongs to a toolsuite called WTC [52] that was designed to prove termination of flowchart programs.

REAVER[7] (REActive system VERifier) is a tool for safety verification of discrete and hybrid systems specified by logico-numerical data-flow languages, like LUSTRE [53]. It features partitioning techniques and logico-numerical analysis methods based on logico-numerical product and power domains (of the APRON [54] and BDDAPRON [55] domain libraries) with convex polyhedra, octagons, intervals, and template polyhedra to abstract numerical state sets. It has frontends for the NBAC format [49], LUSTRE via LUS2NBAC [56], and (subset of) LUCID SYNCHRONE [57]. It implements abstract acceleration for self-loops (§4) including the extension to numerical inputs (§6.1), the logico-numerical technique (§6.3), and numerical backward acceleration with inputs (§6.2). Furthermore, it is able to apply these methods to the analysis of hybrid systems.

[6] `laure.gonnord.org/pro/aspic/aspic.html`

[7] `pop-art.inrialpes.fr/people/schramme/reaver/`

7.2. Experimental Results

In this section, we recall some already published experimental results that show the relevance of the acceleration methods in what concerns precision and efficiency. We also mention applications where abstract acceleration-based invariant generation has been successfully applied, like termination analysis.

Accelerated vs standard LRA. Using ASPIC and its C front-end C2FSM, Feautrier and Gonnord [51] showed the precision of abstract acceleration in comparison to FAST, standard LRA, and its improvements using widening with thresholds (widening "up to" [10]) and guided static analysis [12] on 17 small, but difficult benchmarks[8]. The experiments show that ASPIC generally manages to infer better invariants in 12 cases in comparison to standard LRA, 7 w.r.t. widening "up to", and 9 compared with guided static analysis. With a 15 minutes timeout, FAST is able to compute the exact invariants on 13 benchmarks but is at least 3 times slower.

Proving properties and termination. ASPIC has been used in combination with C2FSM for proving properties in C programs [51] (Table 1).

File	#lines of code	(#variables,#control points)	Proved property
Apache (`simp1_ok`)	30	(5,5)	No buffer Overflow (c2fsm)
Sendmail (`inner_ok`)	32	(4,3)	No buffer Overflow (c2fsm)
Sendmail (`mime_fromqp_arr_ok.c`)	84	(20,20)	No buffer Overflow (aspic)
Spam (`loop_ok`)	42	(8,10)	No Buffer Overflow (aspic)
OpenSER (`parse_config_ok`)	72	(7,30)	No Buffer Overflow (aspic+accel)
`list.c`	38	(20,10)	AssertOK (aspic+delay4+accel)
`disj_simple.c`	13	(4,5)	AssertOK (aspic+accel)
Heapsort (`realheapsort`)	59	(25,55)	Termination (aspic)
Loops (`nestedLoop`)	24	(6,6)	Termination (aspic+delay4+accel)

Table 1: C benchmarks (cf. [58] and [59])

The two first properties were proved within C2FSM (bad states were proved to be unreachable while generating the automaton). Standard linear relation analysis was sufficient to prove three more properties, but the precision gain with acceleration was absolutely mandatory for the rest of the experiments of this table. In some cases we were obliged to combine (abstract) acceleration and delayed widening to obtain the desired precision.

For termination analysis, ASPIC was used as invariant generator inside the WTC toolsuite[9]. The idea is to approximate the search space domain for finding *linear ranking functions* and thus proving termination [52]. The experiments show that in a few cases (7 over 35), abstract acceleration gained enough precision to conclude for termination. In 8 more cases, standard analysis with widening "up to" was precise enough to prove termination, although acceleration gave better results in terms of invariant precision. In all other cases, the obtained invariants were the same.

[8] All these benchmarks can be found on `http://laure.gonnord.org/pro/aspic/aspic.html`

[9] `http://compsys-tools.ens-lyon.fr/rank/index.php`

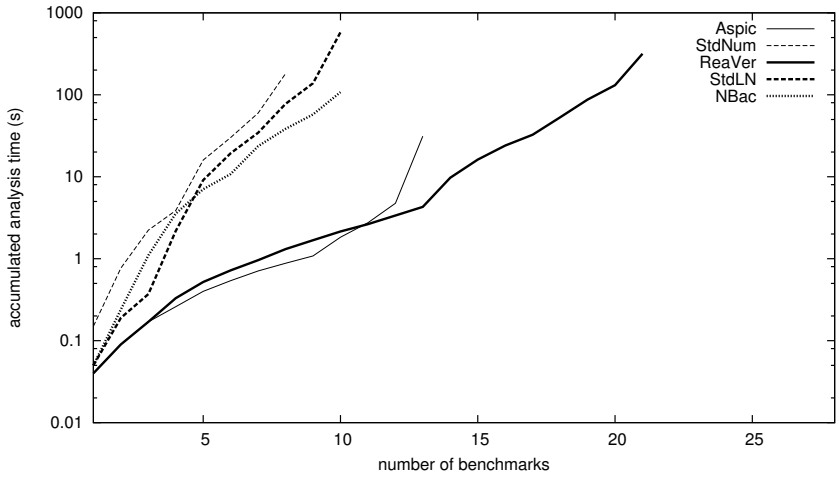


Figure 11: Comparison between ASPIC, standard numerical linear relation analysis (StdNum), REAVER, standard logico-numerical analysis (StdLN), and NBAC.

Reactive system analysis. Fig. 11 depicts the results of an experimental comparison (cf. [50]) on a set of reactive system benchmarks[10] written in LUSTRE. On the one hand, it compares standard analysis methods (standard LRA (§2.1) and NBAC [49]) with abstract acceleration (ASPIC and REAVER), and on the other hand, it compares purely numerical methods (ASPIC and standard numerical LRA) that encode the Boolean states in CFG locations[11] with logico-numerical methods (REAVER, NBAC, and standard logico-numerical LRA) that use state space partitioning heuristics to scale up. It shows that the abstract acceleration-based tools outperform the other methods. This is due to both the increased precision and the faster convergence to a fixed point. Moreover, it shows that REAVER that implements logico-numerical abstract acceleration scales better than the purely numerical ASPIC.

8. Discussion and Related Work

Methods to improve widening. As explained in §2.1.3, the motivation for abstract acceleration was the development of a precise, monotonic widening operator for a specific class of loops.

Delaying widening (parameter N in §2.1.3) helps discovering a better estimate before applying extrapolation. This has an important effect on precision. However, delaying widening in polyhedral analyses is expensive, because coefficients quickly become huge. Remark that delaying widening is not equivalent to loop unrolling, because the convex union is taken after each iteration, whereas loop unrolling creates separate locations.

In systems with nested loops (or loops with loop phases (branches within the loop)), the standard approach often suffers from the fact that the result of the widening sequence is already a (bad) fixed point, and hence the descending sequence cannot recover any information [60]. This

[10] Production line benchmarks available on: `pop-art.inrialpes.fr/people/schramme/reaver/`.

[11] Only the Boolean states that are reachable in the Boolean skeleton of the program are actually enumerated.

typically happens due to transitions that keep "injecting" bad extrapolations during the descending sequence.

Halbwachs [10] observes that distinguishing loop phases can be exploited to obtain a better precision. *Guided static analysis with lookahead widening* [61, 12] follows this idea by alternating ascending and descending sequences on a strictly increasing, finite sequence of restrictions of the program (by adding new transitions) which converges towards the original program. In many cases this method improves the precision, but it ultimately relies on the effectiveness of the descending iterations.

Widening with thresholds tries to limit widening by a given set of constraints $\mathscr{T}$ [10, 62] in order to avoid bad extrapolations in the first place. To achieve this, the result of the standard widening is intersected with those constraints in $\mathscr{T}$ that are satisfied by both arguments of the widening operator. The problem is, though, how to find a set of relevant threshold constraints. A *static threshold inference* method based on propagating the post-condition of a loop guard to the widening points of the CFG is proposed in [28]. *Dynamic threshold inference* methods are for example *counterexample-refined widening* [63], which is based on an (under-approximating) backward analysis, and *widening with landmarks* [11], which extrapolates threshold constraints by estimating the number of loop iterations until the guard of the loop is violated.

A framework for designing new widening operators was proposed in [64, 65]. The proposition for a new widening operator for polyhedron gives better precision locally but as widening is not monotonic, there is no result on the global precision improvement. Most experimental results show an improvement, but there are also counter-examples. Moreover, the cost of the analysis can increase significantly, because the convergence is generally slower.

Abstract acceleration vs. Kleene iteration. Abstract acceleration aims at computing a tight over-approximation of $\alpha(\bigcup_{k\geqslant 0}\tau^k(X_0))$ where X_0 is a convex polyhedron and τ is an affine transformation with an affine guard. Since convex polyhedra are closed under affine transformations ($\alpha(\tau(X_0))=\tau(X_0)$), we have $\alpha(\bigcup_{k\geqslant 0}\tau^k(X_0))=\bigsqcup_{k\geqslant 0}\tau^k(X_0)$. The latter formula is known as *Merge-Over-All-Paths* (MOP) solution of the reachability problem [66], which computes the limit of the sequence:

$$X_0 \qquad X_1 = X_0 \sqcup \tau(X_0) \qquad X_2 = X_0 \sqcup \tau(X_0) \sqcup \tau^2(X_0) \qquad \ldots$$

In contrast, the standard approach in abstract interpretation computes the fixed point X'_∞ of $X = X_0 \sqcup \tau(X)$, known as the *Minimal-Fixed-Point* (MFP) solution. It proceeds as follows:

$$X'_0 = X_0 \qquad X'_1 = X'_0 \sqcup \tau(X'_0) \qquad X'_2 = X'_0 \sqcup \tau(X'_0 \sqcup \tau(X'_0)) \qquad \ldots$$

The MOP solution is more precise than the MFP solution [66]. The reason is that, in general, τ does not distribute over $\sqcup$ and we have $\tau(X_1)\sqcup\tau(X_2)\sqsubseteq\tau(X_1\sqcup X_2)$. For instance, if $X_0=[0,0]$ and $\tau : x\leqslant 1 \rightarrow x'=x+2$, we have $X_2=[0,0]\sqcup[2,2]\sqcup\bot=[0,2]$ and $X'_2=[0,0]\sqcup\tau([0,2])=[0,3]$. Since abstract acceleration should yield a tight over-approximation of the MOP solution, we should generally have the relationship $\bigsqcup_{k\geqslant 0}\tau^k(X_0)\sqsubseteq\tau^\otimes(X_0)\sqsubseteq X'_\infty$, *i.e.*, abstract acceleration should be more precise than the standard abstract interpretation approach (MFP) even when assuming convergence of the Kleene iteration without widening.

Abstract acceleration in non-flat systems. Generally, the invariants computed by abstract acceleration of a single self-loop τ are not inductive, which implies that $\tau^\otimes$ is not idempotent, *i.e.*,

$\tau^{\otimes}(\tau^{\otimes}(X_0)) \neq \tau^{\otimes}(X_0)$. While this is not a problem for flat systems, it has negative effects in the presence of nested loops. For example, in the system $(id \circ \tau^*)^*$ we can apply abstract acceleration to the innermost loop: $(id \circ \tau^{\otimes})^* = (\tau^{\otimes})^*$. If $\tau^{\otimes}$ is not idempotent, then the outer loop might not converge and thus widening is needed. Thus, the considerations w.r.t. MOP and MFP solutions for τ^* above apply to $(\tau^{\otimes})^*$ in the same manner. Since this problem arises in particular when the initial set is not contained in the guard G (cf. [67, 47]), Leroux and Sutre [67] propose to accelerate translations by the formula $\tau^{\otimes}(X) = \tau(X \nearrow D)$, *i.e.*, without initially intersecting with G, which is idempotent and hence convergence without widening can be expected more often. However, this formula is clearly less precise and should not be used for accelerating non-nested loops.

Affine derivative closure algorithm. The initial source of inspiration for abstract acceleration was the *affine derivative closure algorithm* of Ancourt et al. [68] which is based on computing an abstract transformer, *i.e.*, a relation between variables $\boldsymbol{x}$ and $\boldsymbol{x}'$, independently of the initial state of the system. The abstract transformer abstracts the effect of the loop by a *polyhedral translation*

$$\mathit{true} \rightarrow \boldsymbol{x}' = \boldsymbol{x} + D_R \quad \text{with} \quad D_R = \{\boldsymbol{d} \mid \exists \boldsymbol{x}, \boldsymbol{\xi}, \boldsymbol{x}' : R(\boldsymbol{x}, \boldsymbol{\xi}, \boldsymbol{x}') \wedge \boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{d}\}$$

where R is the concrete transition relation. The polyhedron D_R is called the "derivative" of the relation R. The effect of multiple loops with relations $R_1, \ldots, R_k$ is abstracted by considering the convex union $\bigsqcup_i D_{R_i}$. Then, the reflexive and transitive closure $R^* = \{(\boldsymbol{x}, \boldsymbol{x}') \mid \exists k \geqslant 0 : \boldsymbol{x}' = \boldsymbol{x} + k\boldsymbol{d} \wedge D_R(\boldsymbol{d})\}$ is applied to a polyhedron X of initial states: $R^*(X) = \{\boldsymbol{x}' \mid \exists \boldsymbol{x} : R^*(\boldsymbol{x}, \boldsymbol{x}') \wedge X(\boldsymbol{x})\}$ The final result is obtained by computing one descending iteration (which eventually takes into account the guards of the loops), in the same way as it is done in standard abstract interpretation after widening. The affine derivative closure algorithm is implemented in the code optimization tool PIPS[12].

In single self-loops with translations or translations/resets, the method works similarly to abstract acceleration, and is expected to yield the same results: although resets cannot be expressed as polyhedral translations (for instance, if $R(x, x') = (x' = 0)$, then $D_R = \{d \mid \exists x, x' : x' = 0 \wedge x' = x + d\} = \top$), this information is recovered during the descending iteration. However, in nested loop situations, these descending iterations may fail (like in Ex. 7). Hence, even though the derivative closure method elegantly deals with multiple loops by taking the convex union of the derivatives, it is also less precise than abstract acceleration for such programs. However, the main advantage of the derivative closure method is that it is more general than abstract acceleration, because it automatically approximates any kind of transformations. Moreover, since it computes abstract transformers, it is modular and can be used in the context of interprocedural analyses.

In the context of applying Presburger-based acceleration to program parallelization, Beletska et al. [69, 70] propose a similar computation scheme for multiple loops that accelerates first the loop bodies and takes into account the guards afterwards, thus trading precision for efficiency.

Strategy iteration. Another approach to overcome the widening issue is to restrict the domain (*e.g.*, using template polyhedra [71] or intervals [72]) and use fixed point iteration strategies that do not

[12] `pips4u.org`

need widening at all. Strategy (policy) iteration methods [73, 74, 75] solve the fixed point equation associated with the reachability analysis by iteratively approximating the least fixed point by fixed points of simpler semantic equations that can be solved exactly using mathematical programming, for example. Strategy iteration methods are able to "accelerate" globally the whole transition system regardless of the graph structure or type of affine transformation, and they effectively compute the least fixed point. However, this is only possible on the simpler domain of template polyhedra. In contrast, abstract acceleration is only able to accelerate some cases of self-loops and cycles with certain types of affine transformations, and it relies on widening in the general case. However, due to the use of general convex polyhedra, it is able to "discover" complex invariant constraints.

Presburger-based acceleration. Most of the related work has been discussed in §2.2. It remains to mention that it has been shown that the transitive closure of difference [15] and octagonal *relations* [76] is also Presburger-definable. A performant algorithm for accelerating such relations is implemented in the tool FLATA [77]. In the most recent work [78], the CEGAR method is extended with an acceleration-based counter example generalization.

Ellipsoidal techniques. Alternative techniques for linear loops originate from the analysis of linear filters frequently found in control systems. These techniques consider *ellipsoidal domains* [79, 80, 81] that are known to be well suited for analyzing systems that are *Lyapunov-stable* (eigenvalues with modulus < 1.), and hence, they target a different class of linear transformations.

Recurrence equations. Kovacs [82] infers loop invariants by *solving the recurrence equations* representing the loop body in closed form. The technique targets exact invariant generation for single loops that admit polynomial equality invariants. While polynomials go beyond the scope of LRA they cannot express all finite monoid transitions (counterexample: rotations). Hence, the class of inferred properties and programs is incomparable to that considered by abstract acceleration. Regardless of performance considerations, the technique could provide a significant enhancement of abstract acceleration to handle a larger class of loops. However, it is not clear how to integrate the technique into LRA.

Quantifier elimination. Acceleration can be viewed as a *loop summarization* method, in the sense that it aims at finding a relation between the states X entering the loop and any future states X' encountered at the loop head (the intersection with the negated loop guard yields the exit states). Symbolically computing such a transformer $R\boldsymbol{x},\boldsymbol{x}' = \exists k \geqslant 0 : \boldsymbol{x}' = \tau^k(\boldsymbol{x})$ is essentially a quantifier elimination problem. For example, Tiwari et al. [83, 84] propose to compute such transformers for linear differential equations $d\boldsymbol{x}(t)/dt = A\boldsymbol{x}(t)$ in the context of hybrid systems, which is a problem similar to the acceleration of discrete linear loops. The transformers are computed using off-the-shelf quantifier elimination tools over real arithmetic, like REDLOG. Although efficiency drawbacks are reported, they are able to solve many tricky benchmarks. Real quantifier elimination is also used by Colón et al. [85, 86] who proposed methods that solve for the coefficients of the linear invariant directly using Farkas' Lemma. Gulwani, Srivastava and Tiwari [87, 88] propose to compute template-based invariants by translating the program into a big $\exists\forall$ formula. Then rational numbers are mapped to bounded-range integers and the formula is solved using bit-blasting or other SMT-based techniques.

9. Summary and Prospects

Abstract acceleration is complement to widening in linear relation analysis. Due to its monotonicity property, it is possible to accelerate the innermost loops precisely while using widening for the outer loops in nested loop situations. Thus, better invariants are computed for programs where a lot of information is lost when using widening only. We have shown how (linear) abstract accelerations are derived for linearly accelerable linear transitions, and we have given experimental results using the tools ASPIC and REAVER.

Open problems and prospects. Abstract acceleration reduces more general linear transformations to some base cases (translations, resets, a.s.o) via a change of basis. This basis change requires the computation of the eigenvalues of the transformation matrix which corresponds to finding the roots of matrix's characteristic polynomial. This is a hard problem and, in general, only numerical approximations can be computed (see, *e.g.*, [89]). However, for our purpose, numerical approximations are not useful, because, for example, the matrices $\begin{pmatrix}1 & 1\\ 0 & 1\end{pmatrix}$ and $\begin{pmatrix}1 & 1\\ 0 & 1+\varepsilon\end{pmatrix}$ are close to each other, but the Jordan form of the first one is itself, whereas the second one is diagonalizable for $\varepsilon > 0$ with the Jordan form $\begin{pmatrix}1 & 0\\ 0 & 1+\varepsilon\end{pmatrix}$, and hence, we could accelerate the first but not the second one. For these reasons, our tools do not yet perform a basis change and accelerate only transitions that are accelerable in the given basis, which is often the case in practical programs. Nonetheless, in order to fully exploit abstract acceleration, we could consider the use of *computer algebra systems* like SAGE[13] that allow us to manipulate rational and algebraic numbers in an exact manner, hoping that the performance is sufficient for most practical cases.

Another direction for future research could be to detect the maximal linearly accelerable subtransformations within a general linear transformation or even in a general transition function. Then, the transition function can be decomposed w.r.t. the accelerable dimensions which are treated using abstract acceleration and the non-accelerable dimensions which can be handled with widening or the derivative closure technique.

At last, it would be of great interest to generalize the abstract acceleration concept to not linearly accelerable (linear) transformations: for example, to compute a reasonably simple, but precise, convex, polyhedral over-approximation of the transitive closure of the self-loop $x \leqslant 10 \rightarrow x' = 2x$ which has an exponential trajectory as a function of time. Computing precise approximations of such behavior is also of high importance in the analysis of hybrid systems, of which the time-continuous behavior is often specified by linear dynamics $\dot{\boldsymbol{x}} = \mathbf{C}\boldsymbol{x}$.

Regarding implemention, we plan to implement abstract acceleration inside the PAGAI[14] tool, which already implements path focusing (§5.3) in a more general framework than ASPIC [44]. This implementation will enable the combination of acceleration, precise fixpoint iteration and linear relation analysis for general C programs. Moreover, we plan to add front-ends for the Numerical Transition System format [90] to our tools in order to favor further experimental comparisons.

[13] `www.sagemath.org`

[14] `pagai.forge.imag.fr`

Acknowledgements We would like to thank Bertrand Jeannet and Nicolas Halbwachs for their valuable comments on this work and on the manuscript.

[1] P. Cousot, N. Halbwachs, Automatic discovery of linear restraints among variables of a program, in: Principles of Programming Languages, ACM, 1978, pp. 84–97.

[2] N. Halbwachs, Détermination automatique de relations linéaires vérifiées par les variables d'un programme, Thèse du 3ème cycle, Institut National Polytechnique de Grenoble (Mar 1979).

[3] P. Cousot, R. Cousot, Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: Principles of Programming Languages, 1977, pp. 238–252.

[4] F. Irigoin, P. Jouvelot, R. Triolet, Semantical interprocedural parallelization: An overview of the PIPS project, in: ACM Int. Conf. on Supercomputing, ICS'91, Köln, 1991, pp. 244–251.

[5] N. Halbwachs, Y.-E. Proy, P. Roumanoff, Verification of real-time systems using linear relation analysis, Formal Methods in System Design 11 (2) (1997) 157–185.

[6] T. A. Henzinger, P.-H. Ho, H. Wong-Toi, HyTech: A model checker for hybrid systems, Journal on Software Tools for Technology Transfer 1 (1-2) (1997) 110–122.

[7] N. Dor, M. Rodeh, M. Sagiv, Cleanness checking of string manipulations in C programs via integer analysis, in: Static Analysis Symposium, Vol. 2126 of LNCS, Springer, 2001, pp. 194–212.

[8] N. Bjorner, I. A. Browne, Z. Manna, Automatic generation of invariants and intermediate assertions, Theoretical Computer Science 173 (1) (1997) 49–87.

[9] P. Roux, R. Delmas, P.-L. Garoche, SMT-AI: an abstract interpreter as oracle for k-induction, ENTCS 267 (2) (2010) 55–68.

[10] N. Halbwachs, Delay analysis in synchronous programs, in: Computer-Aided Verification, Vol. 697 of LNCS, Springer, 1993, pp. 333–346.

[11] A. Simon, A. King, Widening polyhedra with landmarks, in: Asian Symposium on Programming Languages and Systems, Vol. 4279 of LNCS, Springer, 2006, pp. 166–182.

[12] D. Gopan, T. W. Reps, Guided static analysis, in: Static Analysis Symposium, Vol. 4634 of LNCS, Springer, 2007, pp. 349–365.

[13] B. Boigelot, P. Wolper, Symbolic verification with periodic sets, in: Computer-Aided Verification, Vol. 818 of LNCS, Springer, 1994, pp. 55–67.

[14] P. Wolper, B. Boigelot, Verifying systems with infinite but regular state spaces, in: Computer-Aided Verification, Vol. 1427 of LNCS, Springer, 1998, pp. 88–97.

[15] H. Comon, Y. Jurski, Multiple counters automata, safety analysis and Presburger arithmetic, in: Computer-Aided Verification, Vol. 1427 of LNCS, Springer, 1998, pp. 268–279.

[16] A. Finkel, G. Sutre, An algorithm constructing the semilinear Post* for 2-dim Reset/Transfer VASS, in: Mathematical Foundations of computer science, Vol. 1893 of LNCS, Springer, 2000, pp. 353–362.

[17] S. Bardin, A. Finkel, J. Leroux, L. Petrucci, Fast: Fast acceleration of symbolic transition systems, in: Computer-Aided Verification, Vol. 2725 of LNCS, Springer, 2003, pp. 118–121.

[18] L. Gonnord, N. Halbwachs, Combining widening and acceleration in linear relation analysis, in: Static Analysis Symposium, Vol. 4134 of LNCS, Springer, 2006, pp. 144–160.

[19] T. Motzkin, H. Raiffa, G. Thompson, R. Thrall, The double description method, in: H.W.Kuhn, A. Tucker (Eds.), Contributions to the theory of games, Vol. 2, Princeton University Press, 1953.

[20] K. Fukuda, A. Prodon, Double description method revisited, in: Combinatorics and Computer Science, Vol. 1120 of LNCS, Springer, 1995, pp. 91–111.

[21] N. V. Chernikova, Algorithm for finding a general formula for the non-negative solutions of a system of linear inequalities, USSR Computational Mathematics and Mathematical Physics 5 (2) (1965) 228–233.

[22] H. L. Verge, A note on Chernikowa's algorithm, Research report 1662, IRISA (1992).

[23] H. P. Williams, Fourier's method of linear programming and its dual, American Mathematical Monthly 93 (1986) 681–695.

[24] P. Cousot, R. Cousot, Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, in: Programming Language Implementation and Logic Programming, 1992, pp. 269–295.

[25] F. Bourdoncle, Efficient chaotic iteration strategies with widenings, in: Formal Methods in Programming and their Applications, Vol. 735 of LNCS, Springer, 1993, pp. 128–141.

[26] L. Gonnord, Accélération abstraite pour l'amélioration de la précision en analyse des relations linéaires, Thèse de doctorat, Université Joseph Fourier, Grenoble (Oct 2007).

[27] L. Gonnord, N. Halbwachs, Abstract acceleration in linear relation analysis, Tech. Rep. TR-2010-10, Verimag Research Report (2010).

[28] L. Lakhdar-Chaouch, B. Jeannet, A. Girault, Widening with thresholds for programs with complex control graphs, in: Automated Technology for Verification and Analysis, Vol. 6996 of LNCS, Springer, 2011, pp. 492–502.

[29] B. Boigelot, P. Godefroid, Symbolic verification of communication protocols with infinite state spaces using QDDs, Formal Methods in System Design 14 (3) (1997) 237–255.

[30] L. Fribourg, H. Olsén, Proving safety properties of infinite state systems by compilation into Presburger arithmetic, in: Concurrency Theory, Vol. 1243 of LNCS, Springer, 1997, pp. 213–227.

[31] B. Boigelot, Symbolic methods for exploring infinite state spaces, Ph.D. thesis, University of Liège (1998).

[32] S. Bardin, A. Finkel, J. Leroux, P. Schnoebelen, Flat acceleration in symbolic model checking, in: Automated Technology for Verification and Analysis, Vol. 3707 of LNCS, Springer, 2005, pp. 474–488.

[33] M. Presburger, Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt, in: Sprawozdanie z I Kongresu matematyków krajów s lowiańskich, Warszawa 1929, 1930, pp. 92–101.

[34] M. L. Minsky, Computation: Finite and Infinite Machines, Prentice Hall, 1967.

[35] W. Kelly, W. Pugh, E. Rosser, T. Shpeisman, Transitive closure of infinite graphs and its applications, International Journal of Parallel Programming 24 (6) (1996) 579–598.

[36] A. Finkel, J. Leroux, How to compose Presburger-accelerations: Applications to broadcast protocols, in: Foundations of Software Technology and Theoretical Computer Science, Vol. 2556 of LNCS, Springer, 2002, pp. 145–156.

[37] J. Leroux, G. Sutre, Flat counter automata almost everywhere!, in: Automated Technology for Verification and Analysis, Vol. 3707 of LNCS, Springer, 2005, pp. 489–503.

[38] J. Leroux, Algorithmique de la vérification des systèmes à compteurs – approximation et accélération – implémentation dans l'outil FAST, Thèse de doctorat, École Normale Supérieure de Cachan (Dec 2003).

[39] S. Bardin, A. Finkel, J. Leroux, FASTer acceleration of counter automata in practice, in: Tools and Algorithms for the Construction and Analysis of Systems, Vol. 2988 of LNCS, 2004, pp. 576–590.

[40] S. Bardin, A. Finkel, J. Leroux, L. Petrucci, FAST: acceleration from theory to practice, Journal on Software Tools for Technology Transfer 10 (5) (2008) 401–424.

[41] P. Lancaster, M. Tismenetsky, The Theory of Matrices, 2nd Edition, Academic Press, 1984.

[42] P. Schrammel, Logico-numerical verification methods for discrete and hybrid systems, Phd thesis, University of Grenoble (2012).

[43] E. Asarin, O. Maler, A. Pnueli, Reachability analysis of dynamical systems having piecewise-constant derivatives, Theoretical Computer Science 138 (1) (1995) 35–65.

[44] J. Henry, D. Monniaux, M. Moy, Succinct representations for abstract interpretation, in: Static Analysis Symposium, Vol. 7460 of LNCS, Springer, 2012, pp. 283–299.

[45] D. Monniaux, L. Gonnord, Using bounded model checking to focus fixpoint iterations, in: Static Analysis Symposium, LNCS, Springer, 2011, pp. 369–385.

[46] P. Schrammel, B. Jeannet, Extending abstract acceleration to data-flow programs with numerical inputs, in: Numerical and Symbolic Abstract Domains, Vol. 267 of ENTCS, Elsevier, 2010, pp. 101–114.

[47] P. Schrammel, B. Jeannet, Applying abstract acceleration to (co-)reachability analysis of reactive programs, Journal of Symbolic Computation 47 (12) (2012) 1512–1532.

[48] N. Halbwachs, P. Caspi, P. Raymond, D. Pilaud, The synchronous dataflow programming language LUSTRE, Proceedings of the IEEE 79 (9) (1991) 1305–1320.

[49] B. Jeannet, Dynamic partitioning in linear relation analysis. application to the verification of reactive systems, Formal Methods in System Design 23 (1) (2003) 5–37.

[50] P. Schrammel, B. Jeannet, Logico-numerical abstract acceleration and application to the verification of data-flow programs, in: Static Analysis Symposium, Vol. 6887 of LNCS, Springer, 2011, pp. 233–248.

[51] P. Feautrier, L. Gonnord, Accelerated Invariant Generation for C Programs with Aspic and C2fsm, in: Tools for Automatic Program Analysis, 2010, pp. 3–13.

[52] C. Alias, A. Darte, P. Feautrier, L. Gonnord, Multi-dimensional Rankings, Program Termination, and Complexity Bounds of Flowchart Programs, in: Static Analysis Symposium, Vol. 6337 of LNCS, Springer, 2010, pp. 117–133.

[53] P. Caspi, D. Pilaud, N. Halbwachs, J. A. Plaice, LUSTRE: a declarative language for real-time programming, in: Principles of Programming Languages, ACM, 1987, pp. 178–188.

[54] B. Jeannet, A. Miné, APRON: A library of numerical abstract domains for static analysis, in: Computer-Aided Verification, Vol. 5643 of LNCS, Springer, 2009, pp. 661–667.

[55] B. Jeannet, Some experience on the software engineering of abstract interpretation tools, in: Tools for Automatic Program Analysis, Vol. 267 of ENTCS, Elsevier, 2010.

[56] B. Jeannet, Partitionnement dynamique dans l'analyse de relations linéaires et application à la vérification de programmes synchrones, Thèse de doctorat, Grenoble INP (Sept 2000).

[57] P. Caspi, G. Hamon, M. Pouzet, Synchronous functional programming: The Lucid Synchrone experiment, in: N. Navet, S. Merz (Eds.), Real-Time Systems: Description and Verification Techniques: Theory and Tools, Hermes, 2008.

[58] K. Ku, T. Hart, M. Chechik, D. Lie, A buffer overflow benchmark for software model checkers, in: Automated Software Engineering, IEEE/ACM, 2007, pp. 389–392.

[59] A. Gupta, A. Rybalchenko, InvGen: An efficient invariant generator, in: Computer-Aided Verification, 2009, pp. 634–640.

[60] N. Halbwachs, J. Henry, When the decreasing sequence fails, in: Static Analysis Symposium, Vol. 7460 of LNCS, Springer, 2012, pp. 198–213.

[61] D. Gopan, T. W. Reps, Lookahead widening, in: Computer-Aided Verification, Vol. 4144 of LNCS, Springer, 2006, pp. 452–466.

[62] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, X. Rival, Why does ASTRÉE scale up?, Formal Methods in System Design 35 (3) (2009) 229–264.

[63] C. Wang, Z. Yang, A. Gupta, F. Ivancic, Using counterexamples for improving the precision of reachability computation with polyhedra, in: Computer-Aided Verification, Vol. 4590 of LNCS, Springer, 2007, pp. 352–365.

[64] R. Bagnara, P. M. Hill, E. Ricci, E. Zaffanella, Precise widening operators for convex polyhedra, in: Static Analysis Symposium, Vol. 2694 of LNCS, Springer, 2003, pp. 337–354.

[65] R. Bagnara, P. M. Hill, E. Zaffanella, Widening operators for powerset domains, in: Verification, Model Checking, and Abstract Interpretation, Vol. 2937 of LNCS, Springer, 2004, pp. 135–148.

[66] J. B. Kam, J. D. Ullman, Monotone data flow analysis frameworks, Acta Informatica 7 (1977) 305–317.

[67] J. Leroux, G. Sutre, Acceleration in convex data-flow analysis, in: Foundations of Software Technology and Theoretical Computer Science, Vol. 4855 of LNCS, Springer, 2007, pp. 520–531.

[68] C. Ancourt, F. Coelho, F. Irigoin, A modular static analysis approach to affine loop invariants detection, in: Numerical and Symbolic Abstract Domains, Vol. 267 of ENTCS, Elsevier, 2010, pp. 3–16.

[69] A. Beletska, D. Barthou, W. Bielecki, A. Cohen, Computing the transitive closure of a union of affine integer tuple relations, in: Combinatorial Optimization and Applications, Vol. 6508 of LNCS, Springer, 2009, pp. 98–109.

[70] S. Verdoolaege, A. Cohen, A. Beletska, Transitive closures of affine integer tuple relations and their overapproximations, in: Static Analysis Symposium, Vol. 6887 of LNCS, Springer, 2011, pp. 216–232.

[71] S. Sankaranarayanan, H. B. Sipma, Z. Manna, Scalable analysis of linear systems using mathematical programming, in: Verification, Model Checking, and Abstract Interpretation, Vol. 3385 of LNCS, Springer, 2005, pp. 25–41.

[72] Z. Su, D. Wagner, A class of polynomially solvable range constraints for interval analysis without widenings and narrowings, in: Tools and Algorithms for the Construction and Analysis of Systems, Vol. 2988 of LNCS, Springer, 2004, pp. 280–295.

[73] A. Costan, S. Gaubert, E. Goubault, M. Martel, S. Putot, A policy iteration algorithm for computing fixed points in static analysis of programs, in: Computer-Aided Verification, Vol. 3576 of LNCS, Springer, 2005, pp. 462–475.

[74] T. Gawlitza, H. Seidl, Precise fixpoint computation through strategy iteration, in: ESOP 2007, LNCS 4421, Springer-Verlag, 2007, pp. 300–315.

[75] T. M. Gawlitza, H. Seidl, A. Adjé, S. Gaubert, É. Goubault, Abstract interpretation meets convex optimization, Journal of Symbolic Computation 47 (12) (2012) 1512–1532.

[76] M. Bozga, C. Gîrlea, R. Iosif, Iterating octagons, in: Tools and Algorithms for the Construction and Analysis of Systems, Vol. 5505 of LNCS, Springer, 2009, pp. 337–351.

[77] M. Bozga, R. Iosif, F. Konecný, Fast acceleration of ultimately periodic relations, in: Computer-Aided Verification, Vol. 6174 of LNCS, Springer, 2010, pp. 227–242.

[78] H. Hojjat, R. Iosif, F. Konečný, V. Kuncak, P. Rümmer, Accelerating interpolants, in: Automated Technology for Verification and Analysis, Vol. 7561 of LNCS, Springer, 2012.

[79] A. Kurzhanskiy, P. Varaiya, Ellipsoidal techniques for reachability analysis, in: Hybrid Systems: Computation and Control, Vol. 1790 of LNCS, Springer, 2000, pp. 202–214.

[80] J. Feret, Static analysis of digital filters, in: European Symposium on Programming, Vol. 2986 of LNCS, 2004, pp. 33–48.

[81] P. Roux, R. Jobredeaux, P.-L. Garoche, E. Feron, A generic ellipsoid abstract domain for linear time invariant systems, in: Hybrid Systems: Computation and Control, ACM, 2012, pp. 105–114.

[82] L. Kovács, Invariant generation for p-solvable loops with assignments, in: Computer Science - Theory and Applications, Vol. 5010 of LNCS, 2008, pp. 349–359.

[83] A. Tiwari, Approximate reachability for linear systems, in: Hybrid Systems: Computation and Control, Vol. 2623 of LNCS, Springer, 2003, pp. 514–525.

[84] S. Sankaranarayanan, A. Tiwari, Relational abstractions for continuous and hybrid systems, in: Computer-Aided Verification, Vol. 6806 of LNCS, Springer, 2011, pp. 686–702.

[85] S. Sankaranarayanan, H. B. Sipma, Z. Manna, Constraint-based linear relations analysis, in: SAS, Vol. 3148 of LNCS, Springer, 2004, pp. 53–68.

[86] M. Colón, S. Sankaranarayanan, H. Sipma, Linear invariant generation using non-linear constraint solving, in: Computer-Aided Verification, Vol. 2725 of LNCS, Springer, 2003, pp. 420–432.

[87] S. Srivastava, S. Gulwani, Program verification using templates over predicate abstraction, in: Programming language design and implementation, ACM, 2009, pp. 223–234.

[88] S. Gulwani, A. Tiwari, Constraint-based approach for analysis of hybrid systems, in: Computer-Aided Verification, Vol. 5123 of LNCS, Springer, 2008, pp. 190–203.

[89] J. H. Wilkinson (Ed.), The algebraic eigenvalue problem, Oxford University Press, 1988.

[90] H. Hojjat, F. Konecny, F. Garnier, R. Iosif, V. Kuncak, P. Ruemmer, A verification toolkit for numerical transition systems (tool paper), in: Formal Methods, Vol. 7436 of LNCS, Springer, 2012.

Appendix A. Omitted Proofs

Appendix A.1. Jordan form characterization of linearly accelerable transformations

Lemma 6 (Jordan block of size 1). *A transition $\tau(X) : \boldsymbol{x}' = \mathbf{J}\boldsymbol{x}$ where $\mathbf{J}$ is a Jordan block of size 1 is linearly accelerable iff its associated eigenvalue is either zero or a complex root of unity, i.e., $\lambda \in \{0\} \cup \{e^{i2\pi\frac{q}{p}} \mid p,q \in \mathbb{N}\}$.*

PROOF: A Jordan block of size $m = 1$ consists of its associated eigenvalue: $\mathbf{J} = (\lambda)$. The linear acceleration for τ is thus

$$\tau^*(X) = \bigcup_{k\geqslant 0}\{\lambda^k x \mid x \in X\} = \bigcup_{l}\{a_l x + b_l k \mid k \geqslant 0, x \in X\}$$

This means that we have to look for values of λ such that there is a finite number of values for the coefficients a_l, b_l as solutions of the equation $\forall k \geqslant 0, x \in X : \lambda^k x = a_l x + b_l k$. We distiniguish cases according to k:

- $k = 0$: $\lambda^0 = 1$ for any λ, hence $a_0 = 1, b_0 = 0$.
- $k \geqslant 1$: By writing λ^k in polar form $\rho^k e^{i\theta k}$ we get the solutions

$$\begin{cases} \lambda = 0, & a_0 = 0, & b_0 = 0 & (i.e.,\ 0^k e^{i\theta k} = 0x + 0k) \\ \lambda = e^{i2\pi\frac{q}{p}}, & a_k = e^{i2\pi\frac{q}{p}k}, & b_k = 0, p, q \in \mathbb{N} & (i.e.,\ 1^k e^{i2\pi\frac{q}{p}k} x = e^{i2\pi\frac{q}{p}k} x + 0k). \end{cases}$$

  For the second solution, the number of values for a_k is finite because they are periodic: $e^{i2\pi\frac{q}{p}} = e^{i(2\pi\frac{q}{p} + 2\pi q j)}, j \in \mathbb{Z}$. Hence, there are p distinct values for a_k with $0 \leqslant k \leqslant p-1$.

This yields:

$$\begin{array}{ll} \lambda = 0: & \tau^* = \lambda X.X \cup \{0\} \\ \lambda = e^{i2\pi\frac{q}{p}}: & \tau^* = \lambda X.\bigcup_{0\leqslant l\leqslant p-1}\{\left(e^{i2\pi\frac{q}{p}l} x\right) \mid x \in X\} \end{array}$$

□

Lemma 7 (Jordan block of size 2). *A transition $\tau(X) : \boldsymbol{x}' = \mathbf{J}\boldsymbol{x}$ where $\mathbf{J}$ is a Jordan block of size 2 is linearly accelerable iff its associated eigenvalue is*

– *either zero ($\lambda = 0$) or*
– *a complex root of unity ($\lambda \in \{e^{i2\pi\frac{q}{p}} \mid p,q \in \mathbb{N}\}$) and if in this case the variable associated to the second dimension of the block has only a finite number of values in X.*

PROOF: A Jordan block of size $m = 2$ has the form $\mathbf{J} = \begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix}$. The linear acceleration for τ is

$$\begin{array}{rcl} \tau^*(X) & = & X \cup \bigcup_{k\geqslant 1}\left\{ \begin{pmatrix} \lambda^k & k\lambda^{k-1} \\ 0 & \lambda^k \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \mid \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in X \right\} \\ & = & \bigcup_l \left\{ \begin{pmatrix} a_{l,1} & a_{l,2} \\ a_{l,3} & a_{l,4} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + k \begin{pmatrix} b_{l,1} \\ b_{l,2} \end{pmatrix} \mid k \geqslant 0, \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in X \right\} \end{array}$$

The case $k = 0$ is the identity for any eigenvalue λ, thus, we concentrate on $k \geqslant 1$: We have to find values of λ such that there is a finite number of values for the coefficients $a_{l,\cdot}, b_{l,\cdot}$ in the equation

$$\forall k \geqslant 1, \boldsymbol{x} \in X : \begin{cases} \lambda^k x_1 + k\lambda^{k-1} x_2 & = & a_{l,1}x_1 + a_{l,2}x_2 + b_{l,1}k \\ \lambda^k x_2 & = & a_{l,3}x_1 + a_{l,4}x_2 + b_{l,2}k \end{cases}$$

We distinguish cases by values of k:

- $k=1$: The left-hand side of the first equation reduces to λx_1+x_2. Hence, we can match left and right-hand sides for any values of λ: $a_{0,1}=a_{0,4}=\lambda, a_{0,2}=1$, all other coefficents are 0.
- $k\geqslant 2$: In this case we cannot match $k\lambda^{k-1}x_2$ with $a_{l,2}x_2$, but we can match it with $b_{l,1}k$ under the assumption that x_2 has a finite number of values in X, which gives us (besides $\lambda=0$) the following solution ($p,q\in\mathbb{N}$):

$$\forall k\geqslant 2, \boldsymbol{x}\in X: \begin{cases} \underbrace{e^{i2\pi\frac{q}{p}k}}_{\lambda^k}x_1+k\underbrace{e^{i2\pi\frac{q}{p}(k-1)}}_{\lambda^{k-1}}x_2 &= \underbrace{e^{i2\pi\frac{q}{p}k}}_{a_{kx_2,1}}x_1+\underbrace{0}_{a_{kx_2,2}}\cdot x_2+\underbrace{e^{i2\pi\frac{q}{p}(k-1)}x_2}_{b_{kx_2,1}}k \\ \underbrace{e^{i2\pi\frac{q}{p}k}}_{\lambda^k}x_2 &= \underbrace{0}_{a_{kx_2,3}}\cdot x_1+\underbrace{e^{i2\pi\frac{q}{p}k}}_{a_{kx_2,4}}x_2+\underbrace{0}_{b_{kx_2,2}}\cdot k \end{cases}$$

As in the proof for Lem. 6 there is a finite number of values for $e^{i2\pi\frac{q}{p}k}$ because it is periodic. This yields:

$$\lambda=0: \qquad \tau^* = \lambda X.X\cup\left\{\begin{pmatrix} x_2 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \end{pmatrix}\right\}$$

$$\lambda=e^{i2\pi\frac{q}{p}}: \quad \tau^* = \begin{cases} \lambda X.X\cup \\ \bigcup_{1\leqslant l\leqslant p, x_2}\left\{\begin{pmatrix} e^{i2\pi\frac{q}{p}l}x_1+kle^{i2\pi\frac{q}{p}(l-1)}x_2 \\ e^{i2\pi\frac{q}{p}l}x_2 \end{pmatrix} \mid \boldsymbol{x}\in X, k\geqslant 0\right\} \end{cases}$$

□

Lemma 8 (Jordan block of size >2). *A transition $\tau(X):\boldsymbol{x}'=\mathbf{J}\boldsymbol{x}$ where $\mathbf{J}$ is a Jordan block of size >2 is linearly accelerable iff its associated eigenvalue is zero.*

PROOF: Jordan blocks of size $m>2$ have the form: $\mathbf{J}=\begin{pmatrix} \lambda & 1 & 0 & \dots & 0 \\ 0 & \lambda & 1 & \dots & 0 \\ \dots & \dots & \ddots & \ddots & \dots \\ 0 & \dots & 0 & \lambda & 1 \\ 0 & \dots & 0 & 0 & \lambda \end{pmatrix}$.

Their powers have the form:

$$\mathbf{J}^k=\begin{pmatrix} \lambda^k & C_1^k\lambda^{k-1} & C_2^k\lambda^{k-2} & \dots & C_{m-1}^k\lambda^{k-m+1} \\ 0 & \lambda^k & C_1^k\lambda^{k-1} & \dots & C_{m-2}^k\lambda^{k-m+2} \\ \dots & \dots & \ddots & \ddots & \dots \\ 0 & \dots & 0 & \lambda^k & C_1^k\lambda^{k-1} \\ 0 & \dots & 0 & 0 & \lambda^k \end{pmatrix}$$

where C_j^k are the binomial coefficients.

For any value of $\lambda\neq 0$ we have polynomials in k of order m, hence we cannot match the coefficients with a linear form in k.

Thus, we have only $\lambda=0$: $\tau^*=\lambda X.\bigcup_{0\leqslant k\leqslant m}\{\mathbf{J}^k\boldsymbol{x}\mid \boldsymbol{x}\in X\}$ □

Appendix A.2. Jordan form characterization of finite monoid transformations

Theorem 9 (Jordan form of finite monoid affine transformations). *The Jordan form of the homogeneous transformation matrix $\begin{pmatrix} \mathbf{C} & \boldsymbol{d} \\ \mathbf{0} & 1 \end{pmatrix}$, where $\{\mathbf{C}^k\mid k\geqslant 0\}$ is finite, consists of*

– *Jordan blocks of size 1 with eigenvalues which are complex roots of unity,*
– *at most one block of size 2 with eigenvalue* 1 *where the variable associated with the second dimension is a constant equal to* 1*, and*
– *blocks with eigenvalue* 0 *of any size.*

PROOF: We will show that
(1) extending $\mathbf{C}$ to the homogeneous form adds the eigenvalue 1 to the spectrum. Hence, only the Jordan blocks in the Jordan form of $\mathbf{C}$ associated with an eigenvalue 1 are affected by homogenization, and
(2) the Jordan form of the homogenized matrix has at most one Jordan block of size 2 associated to an eigenvalue 1.

(1) follows from the fact that the characteristic polynomial of the homogeneous form is the characteristic polynomial of $\mathbf{C}$ multiplied by $1-\lambda$:

$$det\begin{pmatrix} \mathbf{C}-\lambda\mathbf{I} & \boldsymbol{d} \\ \mathbf{0} & 1-\lambda \end{pmatrix} = (1-\lambda)\cdot det(\mathbf{C}-\lambda\mathbf{I})$$

(because the left-hand side matrix is triangular). The variable corresponding to the dimension added during homogenization is known to equal the constant 1.

(2) We show that the Jordan form of $\mathbf{C}'$ has at most one Jordan block of size 2 associated with an eigenvalue 1: Assume that the Jordan form of the $(n-1)$-dimensional matrix $\mathbf{C}$ has $m-1$ blocks of size 1 associated with eigenvalue 1. Then, the homogeneous form $\mathbf{C}'$ has
- exactly 1 Jordan block of size 2 (with eigenvalue 1) and $m-2$ blocks of size 1 (with eigenvalue 1) if $ker(\mathbf{C}'-\mathbf{I})$ has dimension $m-1$, *i.e.*, $rank(\mathbf{C}'-\mathbf{I}) = n-m+1$;
- no Jordan block of size 2 (with eigenvalue 1) and m blocks of size 1 (with eigenvalue 1) if $ker(\mathbf{C}'-\mathbf{I})$ has dimension m, *i.e.*, $rank(\mathbf{C}'-\mathbf{I}) = n-m$.

Since the eigenvalues 1 of $\mathbf{C}$ have all geometric multiplicity 1, $\mathbf{C}-\mathbf{I}$ has $n-m$ linearly independent column vectors. Hence, $\mathbf{C}'-\mathbf{I}$ has $n-m$ linearly independent column vectors iff the additional column vector $\begin{pmatrix} \boldsymbol{d} \\ 0 \end{pmatrix}$ is not linearly independent from the others; otherwise it has $n-m+1$ linearly independent column vectors. Hence, we have $n-m \leqslant rank(\mathbf{C}'-\mathbf{I}) \leqslant n-m+1$. □

Example 11 (Jordan form characterization). *Consider* $\mathbf{C} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$ *and* $\boldsymbol{d} = (0,1,2)^T$.
$\mathbf{C}$ *has the eigenvalues* $\{1, -\frac{1}{2} \pm \frac{1}{2}i\sqrt{3}\}$. *Hence, the homogeneous form* $\mathbf{C}'$ *has 4 dimensions* ($n=4$) *and an eigenvalue* 1 *with algebraic multiplicity* $m=2$.
The matrix $\mathbf{C}'-\mathbf{I} = \begin{pmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 1 \\ 1 & 0 & -1 & 2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ *has* $n-m+1=3$ *linearly independent column vectors, hence the Jordan form has one block with size 2 associated with the eigenvalue* 1.
Assume that $\boldsymbol{d}=(-1,0,1)^T$, *then* $\mathbf{C}'-\mathbf{I}$ *would have only* $n-m=2$ *linearly independent column vectors, hence the Jordan form would have 2 blocks of size 1 associated with the eigenvalue* 1.